Logical Methods in Computer Science
Volume 16, Issue 3, 2020, pp. 18:1–18:69
https://lmcs.episciences.org/

Submitted Apr. 19, 2019
Published Sep. 24, 2020

ON RESOLVING NON-DETERMINISM IN CHOREOGRAPHIES

LAURA BOCCHI [a], HERNÁN MELGRATTI [b], AND EMILIO TUOSTO [c]

[a] School of Computing, University of Kent, UK

[b] Instituto de Ciencias de la Computación - Universidad de Buenos Aires - Conicet, Argentina

[c] Gran Sasso Science Institute, IT and Department of Computer Science, University of Leicester, UK

Abstract. Choreographies specify multiparty interactions via message passing. A *realisation* of a choreography is a composition of independent processes that behave as specified by the choreography. Existing relations of correctness/completeness between choreographies and realisations are based on models where choices are non-deterministic. Resolving non-deterministic choices into deterministic choices (e.g., conditional statements) is necessary to correctly characterise the relationship between choreographies and their implementations with concrete programming languages. We introduce a notion of realisability for choreographies –called *whole-spectrum implementation*– where choices are still non-deterministic in choreographies, but are deterministic in their implementations. Our notion of whole spectrum implementation rules out deterministic implementations of roles that, no matter which context they are placed in, will never follow one of the branches of a non-deterministic choice. We give a type discipline for checking whole-spectrum implementations. As a case study, we analyse the POP protocol under the lens of whole-spectrum implementation.

1. Introduction

The context A *choreography* describes the expected interactions of a system in terms of the messages exchanged among its components (aka *roles*):

> *"Using the Web Services Choreography specification, a contract containing a global definition of the common ordering conditions and constraints under which messages are exchanged, is produced [...]. Each party can then use the global definition to build and test solutions that conform to it. The global specification is in turn realised by combination of the resulting local systems [...]"*

The first part of the excerpt above taken from [30] envisages a choreography as a global contract regulating the exchange of messages; the last part identifies a distinctive element of choreographies: the global definition can be used to verify local components (correctly)

Key words and phrases: Choreography, multiparty session types, process algebras, whole-spectrum implementation, message-passing, non-determinism.

Research partly supported by the European Unions Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 778233, by the UBACyT projects 20020170100544BA and 20020170100086BA, and by the PIP project 11220130100148CO.

DOI:10.23638/LMCS-16(3:18)2020

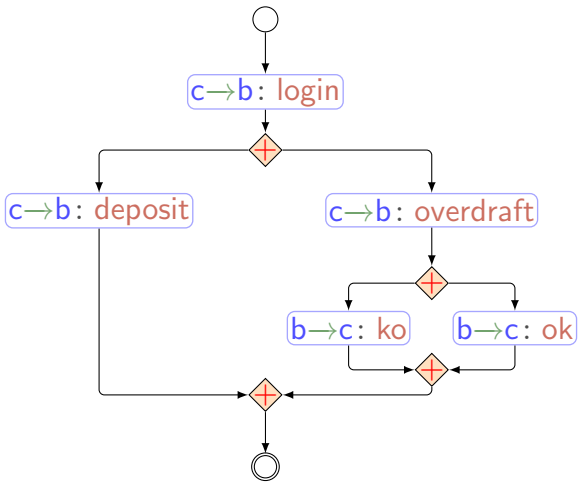


Figure 1: *ATM* Choreography

realise the global contract. A choreography allows for the combination of independently developed distributed components (e.g., services) while hiding implementation details.

Moreover, the communication pattern specified in the choreography yields sufficient information to be projected so to check each component implementing one of the roles. For illustration, take a simple choreography, hereafter called *ATM* and expressed as the global graph [35, 25] on Figure 1, involving a customer c and the cash machine of a bank b. The nodes labelled by ◈ represent branching or merging points of choices. After successful authentication, b offers a deposit and an overdraft service to c. In the global graph this choice corresponds to the topmost ◈ branching over the next interactions between the participants. The bank b can either grant or deny overdrafts asked by the customer c.

On realisations A set of processes is a *realisation* of a choreography when the behaviour emerging from their distributed execution matches the behaviour specified by the choreography. A choreography is *realisable* when it has a realisation.

A realisation of *ATM* can, for example, be given using two CCS-like processes [38] (augmented with internal $\_ \oplus \_$ and external $\_ + \_$ choice operators) for roles b and c:

$$\mathsf{T_b} = \mathsf{login}.(\mathsf{deposit} \;+\; \mathsf{overdraft}.(\overline{\mathsf{ok}} \;\oplus\; \overline{\mathsf{ko}}))$$

$$\mathsf{T_c} = \overline{\mathsf{login}}.(\overline{\mathsf{deposit}} \;\oplus\; \overline{\mathsf{overdraft}}.(\mathsf{ok} \;+\; \mathsf{ko}))$$

In words, $\mathsf{T_b}$ specifies that, after c logs in, b waits to interact either on deposit or on overdraft; in the latter case, b non-deterministically decides whether to grant or deny the overdraft; $\mathsf{T_c}$ is the dual of $\mathsf{T_b}$. Note that *ATM* uses non-determinism to avoid specifying the criteria for b to grant or deny an overdraft. The use of non-determinism is also reflected in realisations, in fact $\mathsf{T_b}$ uses the internal choice operator $\_ \oplus \_$ to model the reaction when c requests an overdraft.

Choreographies can be interpreted either as *constraints* or as *obligations* of distributed interactions [37]. The former interpretation (aka *partial* [37] or weak [43]) admits a realisation if it exhibits a subset of the behaviour. We propose WSI as a criterion to assess if an implementation takes into account all the execution cases prescribed by a choreography. The theory can be used in practice, to implement a static verification checker targeted at specific

languages, and possibly embedded in a (behavioural) type checker (e.g., for Java [27] or Haskel [39]), or an API generation tool (e.g., for Java [28] or F# [40]). The aforementioned tools are generally targeted at checking soundness of implementations, and checking for WSI would add guarantees of 'completeness'. In this paper, we use a process algebra as an abstraction of implementations, instead of a full fledged programming language, to simplify the presentation and the development of the theory. For instance, take

$$\mathsf{T}'_{\mathsf{b}} = \mathsf{login}.(\mathsf{deposit} \ + \ \mathsf{overdraft}.\overline{\mathsf{ko}})$$

then T'_{b} and T_{c} form a partial realisation of *ATM* where overdraft requests are consistently denied. On the contrary, when interpreting choreographies as obligations, a realisation is admissible if it is able to exhibit *all* interaction sequences (hence such realisations are also referred to as *complete* realisations [37]).

For instance, T_{b} and T_{c} form a complete realisation of *ATM*.

The problem Typically, realisations are non-deterministic specifications; here we explore the problem of resolving their non-determinism. In fact, despite being a valuable abstraction mechanism, non-determinism has to be implemented using deterministic constructs such as conditional statements.

Using again *ATM*, we illustrate that traditional notions of complete realisation are not fully satisfactory. The non-deterministic choice in T_{b} abstracts away from the actual conditions used in implementations to resolve the choice. This permits a bank to adopt different policies depending e.g., on the type of the clients' accounts. Consider the (deterministic) implementations B_1 and B_2 of T_{b} below written as value-passing CCS processes:

$$\begin{array}{lcll} B_i & ::= & \mathsf{login}(c).\,(\mathsf{deposit}(x).Q + \mathsf{overdraft}(x).P_i(c)) & \text{for } i = 1,2 \quad (Q \text{ is immaterial}) \\ P_1(c) & ::= & \mathsf{if}\ \mathit{check}(c)\ \mathsf{then}\ \overline{\mathsf{ok}}\ \mathsf{else}\ \overline{\mathsf{ko}} & \\ P_2(c) & ::= & \overline{\mathsf{ko}} & \end{array}$$

Both B_1 and B_2 expect to receive the login credentials c of a client on channel login. After that, they offer the services deposit and overdraft. The implementations differ in the way they handle overdraft requests, which are respectively defined by $P_1(c)$ and $P_2(c)$. The expression $\mathit{check}(c)$ in $P_1(c)$ deterministically discriminates if the overdraft should be granted depending on the login credentials c provided by the client. Differently, $P_2(c)$ refuses any overdraft request. It is not hard to see that both B_1 and B_2 are suitable implementations of T_{b} in partial realisations of the choreography[1] (as e.g. in [20]).

Conversely, neither B_1 nor B_2 can be used in a complete realisation. This is straightforward for B_2 (unable to interact over ok after receiving an overdraft request), but not so evident for B_1. Depending on the credentials c sent by the customer at login time, $\mathit{check}(c)$ will evaluate either to `true` or to `false`. Therefore, B_1 will execute only one of the branches. This will be the case for any possible deterministic implementation of *ATM*: only one branch will be matched. Consequently, there is not a complete, deterministic realisation for *ATM*.

We prefer B_1 to B_2 arguing that they are not equally appealing when interpreting choreographies as obligations. In fact, B_2 consistently precludes one of the alternatives while B_1 guarantees only one or the other alternative (provided that *check* is not a constant function) depending on the deterministic implementation of the role T_{c}.

[1]For instance, both B_1 and B_2 type-check against T_{b} considered as a session type due to the fact that subtyping for session types [23] is contra-variant with respect to internal choices (and covariant with respect to external choices).

Contributions and synopsis We introduce *whole-spectrum implementation* (WSI), a new interpretation of choreographies as interaction obligations. A WSI of a role r guarantees that, whenever the choreography allows r to make an internal choice, there is a context (i.e., an implementation of the remaining roles) for which (the implementation of) r chooses such alternative. Through the paper, we illustrate the use of WSI to analyse the POP2 protocol (i.e., choreography Section 3, implementation Section 5, and verification Section 7.3).

In the following, we use an elaboration of global types from [34] to model choreographies. Implementations of choreographies are abstracted as systems, which are parallel compositions of processes in an asynchronous calculus. *Whole-spectrum implementation* (WSI) is defined in terms of the denotational semantics of global types and systems. A key point on the characterisation of WSI is the distinction between mandatory and optional behaviour arising from the implementation of loops. The denotational semantics of a global type $\mathcal{G}$ is given by the set $\mathcal{R}(\mathcal{G})$ of traces describing mandatory and optional behaviour (Definition 6.6 in Section 6). The denotation of a system S is also a set $\mathcal{R}(S)$ of traces (Definition 6.4 in Section 6), which however do not discriminate optional behaviour. WSI is defined as a *covering* relation $\Subset$ between the traces of implementations $\mathcal{R}(S)$ and those of global types $\mathcal{R}(\mathcal{G})$ (Definition 6.8, Section 6).

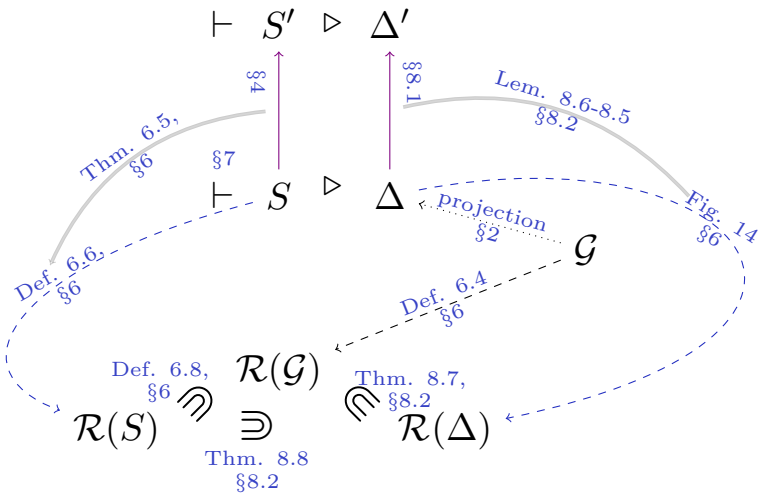
We devise a behavioural typing framework for checking WSI. Our global and local types are in Section 2; the language for implementations is in Section 4. Our typing discipline is introduced in Section 7. As usual, a type judgement $\vdash S \triangleright \Delta$ says that an implementation S has local types Δ, obtained by projecting a global type $\mathcal{G}$ (cf. Section 2). We show a theorem of subject reduction and one of conformance (Theorems 8.2 and 8.4 in Section 8.1) which guarantee that well-typed systems do not deviate from the behaviour specified by their type. Both results rely on the operational semantics of systems (Section 4) and of local types (Section 8.1). We show the adequacy of the denotational semantics of systems to their operational semantics in Theorem 6.5 (Section 6). Analogously, we establish the correspondence between the operational and denotational semantics of local types (Lemmas 8.5 and 8.6, Section 8.2).

The proof that our type system ensures WSI is given in two steps: we prove that

- $\mathcal{R}(\mathcal{G}) \Subset \mathcal{R}(\Delta)$ (Theorem 8.7 in Section 8.2), namely that the traces of the local types projected from a global type $\mathcal{G}$ cover the traces of $\mathcal{G}$; and that
- $\mathcal{R}(\Delta) \Subset \mathcal{R}(S)$ (Theorem 8.8 in Section 8.2), namely that the traces of well-typed systems cover the traces of their local types.

By transitivity of $\Subset$, we obtain $\mathcal{R}(\mathcal{G}) \Subset \mathcal{R}(S)$, which entails WSI for well-typed implementations (Corollary 8.9 in Section 8.2).

The main contributions in this paper can be diagrammatically presented as follows:



This is an extended version of [4]. Besides giving detailed proofs, we simplified some definitions and typing rules. In particular, Section 7.1 was not included in [4] and was added here to remove communication effects from judgements. We also improved the general presentation and refined the running example.

2. Global and Local Types

Our types elaborate from [34] and use a form of iteration, which is more tractable than recursion. We fix three countably infinite and pairwise disjoint sets:

- $\mathbb{U}$ of *shared* names (ranged over by u),
- $\mathbb{Y}$ of *session* channels (ranged over by $\mathsf{y}, \mathsf{s}, \ldots$), and
- $\mathbb{P}$ of *(participants) roles* ranged over by $\mathsf{p}, \mathsf{q}, \mathsf{r}, \ldots$; it is convenient to assume that $\mathbb{P}$ is (isomorphic to) the set of natural numbers.

Basic data types, called *sorts*, (e.g., booleans Bool, integers Int, strings Str, singleton Unit, record types, etc.) are assumed; d ranges over sorts. We use *sorted channels* $\mathsf{y}\ \mathsf{d}$ to specify that channel $\mathsf{y} \in \mathbb{Y}$ is used to exchange data of sort d. We write $\vec{z} = (z_1, \ldots, z_n)$ for a finite sequence of elements $z_1, \ldots, z_n \in Z$ for a given set Z; when no confusion arises, $\vec{z}$ may also denote the set $\{z_1, \ldots, z_n\}$ (e.g., we write $z_2 \in \vec{z}$).

A *global type term* (GTT, for short) G is derived by the following grammar:

$$\mathsf{G} \quad ::= \quad \sum_{i \in I} \mathsf{p} \twoheadrightarrow \mathsf{q_i} : \mathsf{y}_i\ \mathsf{d}_i ; \mathsf{G}_i \qquad \mid \qquad \mathsf{G}; \mathsf{G} \qquad \mid \qquad \mathsf{G}^{*^f} \qquad \mid \qquad \mathsf{end}$$

A GTT $\sum_{i \in I} \mathsf{p} \twoheadrightarrow \mathsf{q_i} : \mathsf{y}_i\ \mathsf{d}_i ; \mathsf{G}_i$ denotes the branching of interactions from a unique selector p to participants q_i for $i \in I \neq \emptyset$; we tacitly assume that $\mathsf{p} \neq \mathsf{q}_i$ for all $i \in I$, and that $\mathsf{q}_i = \mathsf{q}_j$ implies $\mathsf{y}_i \neq \mathsf{y}_j$ for all $i \neq j \in I$ (namely that channels of a same receiver in two different branches are different). For a singleton $I = \{n\}$, $\mathsf{p} \twoheadrightarrow \mathsf{q_n} : \mathsf{y}_n\ \mathsf{d}_n ; \mathsf{G}_n$ shortens $\sum_{i \in I} \mathsf{p} \twoheadrightarrow \mathsf{q_i} : \mathsf{y}_i\ \mathsf{d}_i ; \mathsf{G}_i$. A communication over a channel of sort Unit is a pure synchronisation and usually we write $\mathsf{p} \twoheadrightarrow \mathsf{q} : \mathsf{y}\ ; \mathsf{G}$ instead of $\mathsf{p} \twoheadrightarrow \mathsf{q} : \mathsf{y}\ \mathsf{Unit}; \mathsf{G}$. A GTT can also be the

sequential $(\_;\_)$ composition of two GTTs, the iteration of a GTT $(\_^{*^f})$, or the empty term end. We omit trailing occurrences of end and write $\mathsf{p} \twoheadrightarrow \mathsf{q}\colon \mathsf{y}\ \mathsf{d}$ instead of $\mathsf{p} \twoheadrightarrow \mathsf{q}\colon \mathsf{y}\ \mathsf{d}; \mathsf{end}$.

As in [10], we adopt iteration in global types. Function f in G^{*^f} is an injective map that assigns sorted channels to roles. The mapping is used to indicate how the termination of the iteration is communicated to the roles. More precisely, if $f(\mathsf{p}) = \mathsf{y}\ \mathsf{d}$ then the participant p will receive a message of type d on channel y when the iteration terminates. We use $\mathtt{ch}(f)$ for the set of channels in the image of f, namely

$$\mathtt{ch}(f) \quad = \quad \{\mathsf{y} \mid f(\mathsf{p}) = \mathsf{y}\ \mathsf{d} \quad \text{and} \quad \mathsf{p} \in \mathrm{dom}(f)\}$$

Example 2.1 (Iterative GTT)**.** We revisit the scenario introduced in Section 1, which involves a client c and a bank b. The GTT G below defines a protocol in which c, after being logged in, may perform zero or more deposits and overdrafts.

$$\begin{aligned}\mathsf{G} = \mathsf{c} \twoheadrightarrow \mathsf{b}\colon \mathsf{login}\ \mathsf{Str}; (\ & \\ & \mathsf{c} \twoheadrightarrow \mathsf{b}\colon \mathsf{deposit}\ \mathsf{Int} \\ & + \\ & \mathsf{c} \twoheadrightarrow \mathsf{b}\colon \mathsf{overdraft}\ \mathsf{Int}; (\mathsf{b} \twoheadrightarrow \mathsf{c}\colon \mathsf{ok}\ + \mathsf{b} \twoheadrightarrow \mathsf{c}\colon \mathsf{ko}\ )\quad)^{*^{\mathsf{b}\mapsto\mathsf{quit}}}\end{aligned}$$

The protocol starts with the interaction $\mathsf{c} \twoheadrightarrow \mathsf{b}\colon \mathsf{login}\ \mathsf{Str}$, i.e., c sends to b its login information (of sort Str) over channel login. Then, the protocol continues as an iterative GTT in which c decides to either perform a deposit, ask for an overdraft, or finalise the protocol. The protocol can be finalised by c by sending to b a message (of omitted sort Unit) on channel quit, as indicated by the function $\mathsf{b} \mapsto \mathsf{quit}$ used to decorate the iterative type. In the interaction $\mathsf{c} \twoheadrightarrow \mathsf{b}\colon \mathsf{deposit}\ \mathsf{Int}$, c requests a deposit by sending the amount to be deposited over channel $\mathsf{deposit}$. In the interaction $\mathsf{c} \twoheadrightarrow \mathsf{b}\colon \mathsf{overdraft}\ \mathsf{Int}$, c requests an overdraft. Note that in the case of overdraft request c waits for a notification from b on whether the request is granted or denied (a message over ok or ko, respectively, of the omitted sort Unit). Once the chosen branch has been completed, the iteration can be restarted. ◇

We introduce some useful auxiliary notions. For a GTT G

- the set of *participants* $\mathcal{P}(\mathsf{G})$ *of* G is

$$\begin{aligned}\mathcal{P}(\sum_{i\in I} \mathsf{p} \twoheadrightarrow \mathsf{q}_i\colon \mathsf{y}_i\ \mathsf{d}_i; \mathsf{G}_i) &= \{\mathsf{p}\} \cup \bigcup_{i\in I}(\{\mathsf{q}_i\} \cup \mathcal{P}(\mathsf{G}_i)) \\ \mathcal{P}(\mathsf{G}; \mathsf{G}') &= \mathcal{P}(\mathsf{G}) \cup \mathcal{P}(\mathsf{G}') \\ \mathcal{P}(\mathsf{G}^{*^f}) &= \mathcal{P}(\mathsf{G}) \\ \mathcal{P}(\mathsf{end}) &= \emptyset\end{aligned}$$

- the set of participants ready to send a message, or *ready participants* $\mathtt{rdy}(\mathsf{G})$ *of* G is

$$\begin{aligned}\mathtt{rdy}(\sum_{i\in I} \mathsf{p} \twoheadrightarrow \mathsf{q}_i\colon \mathsf{y}_i\ \mathsf{d}_i; \mathsf{G}_i) &= \{\mathsf{p}\} \\ \mathtt{rdy}(\mathsf{G}; \mathsf{G}') &= \begin{cases}\mathtt{rdy}(\mathsf{G}) & \text{if } \mathtt{rdy}(\mathsf{G}) \neq \emptyset \\ \mathtt{rdy}(\mathsf{G}') & \text{if } \mathtt{rdy}(\mathsf{G}) = \emptyset\end{cases} \\ \mathtt{rdy}(\mathsf{G}^{*^f}) &= \mathtt{rdy}(\mathsf{G}) \\ \mathtt{rdy}(\mathsf{end}) &= \emptyset\end{aligned}$$

- the set of *channel names* $\mathtt{ch}(\mathsf{G}) \subseteq \mathbb{Y}$ *of* G is

$$\begin{array}{rcl} \mathtt{ch}(\sum_{i\in I} \mathsf{p} \twoheadrightarrow \mathsf{q}_i\colon \mathsf{y}_i\ \mathsf{d}_i; \mathsf{G}_i) & = & \bigcup_{i\in I}\{\mathsf{y}_i\} \\ \mathtt{ch}(\mathsf{G};\mathsf{G}') & = & \mathtt{ch}(\mathsf{G}) \cup \mathtt{ch}(\mathsf{G}') \\ \mathtt{ch}(\mathsf{G}^{*^f}) & = & \mathtt{ch}(\mathsf{G}) \cup \mathtt{ch}(f) \\ \mathtt{ch}(\mathsf{end}) & = & \emptyset \end{array}$$

Example 2.2. The set $\mathtt{ch}(\mathsf{G}) = \{\mathsf{login}, \mathsf{deposit}, \mathsf{overdraft}, \mathsf{ok}, \mathsf{ko}, \mathsf{quit}\}$ is the set of channels used by the choreography G in Example 2.1. Also, we have the set of participants $\mathcal{P}(\mathsf{G}) = \{\mathsf{b}, \mathsf{c}\}$, where c is the one that can initiate the interaction, i.e., $\mathtt{rdy}(\mathsf{G}) = \{\mathsf{c}\}$. ◇

A *global type* is defined by an equation

$$\mathcal{G}(\vec{\mathsf{y}}) \triangleq \mathsf{G} \qquad \text{where} \qquad \mathtt{ch}(\mathsf{G}) \subseteq \vec{\mathsf{y}} \subseteq \mathbb{Y} \quad \text{and} \quad \vec{\mathsf{y}} \text{ are pairwise distinct names}$$

We abbreviate $\mathcal{G}(\vec{\mathsf{y}}) \triangleq \mathsf{G}$ with $\mathcal{G}(\vec{\mathsf{y}})$ when G is immaterial; we write $\mathcal{G}$ or G instead of $\mathcal{G}(\vec{\mathsf{y}})$ when parameters are understood.

For technical reasons (*cf.* Section 7), our global types are explicitly parameterised on session channels; however, they will be considered equivalent up-to renaming of parameters. More precisely, let $\equiv_{\mathrm{GTT}}$ be the structural congruence relation on GTTs such that

- $\_;\_$ and $\_ + \_$ form monoids with identity end
- and $\_ + \_$ is commutative;

we say that $\mathcal{G}_1(\vec{\mathsf{y}}) \triangleq \mathsf{G}_1$ and $\mathcal{G}_2(\vec{\mathsf{z}}) \triangleq \mathsf{G}_2$ are structurally equivalent (written $\mathcal{G}_1 \equiv \mathcal{G}_2$) when $\mathsf{G}_1 \equiv_{\mathrm{GTT}} \mathsf{G}_2\{\vec{\mathsf{z}}/\vec{\mathsf{y}}\}$ where, as usual, $\{\vec{\mathsf{y}}/\vec{\mathsf{y}}\}$ is the capture avoiding substitution replacing the i-th element of $\vec{\mathsf{y}}$ with the i-th element of $\vec{\mathsf{z}}$ (for which we assume $\vec{\mathsf{y}}$ to be a tuple of pairwise distinct names of the same length as $\vec{\mathsf{z}}$).

The extensions of $\mathcal{P}(\_)$ and $\mathtt{rdy}(\_)$ to $\mathcal{G}(\vec{\mathsf{y}}) \triangleq \mathsf{G}$ are straightforward: $\mathcal{P}(\mathcal{G}) = \mathcal{P}(\mathsf{G})$ and $\mathtt{rdy}(\mathcal{G}) = \mathtt{rdy}(\mathsf{G})$.

As customary in session types, we restrict our attention to *well-formed* global types in order to rule out specifications that cannot be implemented distributively. We borrow from [26, 10] the standard wellformedness conditions of *knowledge of choice* and *linearity*.

Knowledge of choice requires a unique-selector in a choice and that any other participant can determine the chosen branch from the received messages. For instance, the global type

$$\mathsf{p} \twoheadrightarrow \mathsf{q}\colon \mathsf{y}_1\ ; \mathsf{G}_1 \ + \ \mathsf{p} \twoheadrightarrow \mathsf{r}\colon \mathsf{y}_2\ ; \mathsf{s} \twoheadrightarrow \mathsf{q}\colon \mathsf{y}_1\ ; \mathsf{G}_2$$

violates the knowledge of choice condition: although there is a unique-selector p in the choice (hence the first part of the condition is satisfied), there is a participant q that cannot determine the chosen branch. More precisely, q is ready to receive a message on y_1 either from p in the first branch or from s on the second one; hence q cannot determine which branch has been selected after the input on y_1.

Linearity requires absence of communication races on channels. Races occur when causally unrelated interactions happen on a same channel. Consider

$$\mathsf{p} \twoheadrightarrow \mathsf{q}\colon \mathsf{y}\ ; \mathsf{r} \twoheadrightarrow \mathsf{q}\colon \mathsf{y}$$

where the two sent actions yield a race on y since they are concurrent because performed by different senders. On the contrary,

$$\mathsf{p} \twoheadrightarrow \mathsf{q}\colon \mathsf{y}\ ; \mathsf{q} \twoheadrightarrow \mathsf{r}\colon \mathsf{y}$$

satisfies linearity because the two interactions on y are executed sequentially, hence causally related.

We introduce an additional wellformedness condition (Definition 2.3) that is specific to our form of iteration.

Definition 2.3 (Well-formed iteration)**.** A GTT of the form G^f is a well-formed iteration if:

(1) $\mathtt{ch}(f) \cap \mathtt{ch}(\mathsf{G}) = \emptyset$,
(2) $\mathtt{rdy}(\mathsf{G})$ is a singleton (we call the participant in $\mathtt{rdy}(\mathsf{G})$ the *iteration-controller of* G) and $\mathrm{dom}(f) = \mathcal{P}(\mathsf{G}) \backslash \mathtt{rdy}(\mathsf{G})$,
(3) for any proper subterm $\mathsf{G}_1^{*^{f_1}}$ of G, $\mathtt{ch}(f_1) \cap \mathtt{ch}(f) = \emptyset$.

By condition (1), the channels used to terminate an iteration are disjoint from those in the body G. This is fundamental to avoid confusion when implementing iterations and resembles the condition about knowledge of choices. Consider the global type

$$(\mathsf{p} \twoheadrightarrow \mathsf{q}\colon \mathsf{y}\ ; \mathsf{G}_1)^{*^{\mathsf{q}\mapsto\mathsf{y}}}; \mathsf{G}_2$$

that violates condition (1). Note that after receiving a message from p on y, q is unable to determine whether it should behave as specified by G_1 (i.e., to perform the body of the iteration) or G_2 (i.e., to exit the iteration).

Condition (2) requires a unique role (the iteration-controller) to be the one deciding whether to execute the iteration body or to terminate by notifying all other participants in global type. This condition avoids situations in which a participant is unaware of the fact that some iteration has been finalised. For example, in the following GTT

$$(\mathsf{p} \twoheadrightarrow \mathsf{q}\colon \mathsf{y}_1\ ; \mathsf{q} \twoheadrightarrow \mathsf{r}\colon \mathsf{y}_2\ )^{*^{\mathsf{q}\mapsto\mathsf{y}_3}}; \mathsf{G}_2$$

r will not receive any message when p decides to conclude the iteration.

Finally, condition (3) prevents interference between terminations of nested iterations. Consider

$$((\mathsf{p} \twoheadrightarrow \mathsf{q}\colon \mathsf{y}_1\ )^{*^{\mathsf{q}\mapsto\mathsf{y}_2}}; \mathsf{q} \twoheadrightarrow \mathsf{r}\colon \mathsf{y}_3\ )^{*^{\mathsf{q}\mapsto\mathsf{y}_2,\mathsf{r}\mapsto\mathsf{y}_4}}; \mathsf{G}_2$$

Note that after receiving a message on y_2, q is unable to determine if p has concluded the inner or the outer iteration.

Hereafter, we will assume that for every $\mathcal{G}(\vec{\mathsf{y}}) \triangleq \mathsf{G}$, G satisfies knowledge of choice and linearity (form [26, 10]), and that every iterative GTT appearing in G is a well-formed iteration.

A *local type term* (LTT for short) T is derived from the following grammar:

$$\mathsf{T} \ ::= \ \bigoplus_{i\in I} \overline{\mathsf{y}_i}\ \mathsf{d}_i.\mathsf{T}_i \quad \Big| \quad \sum_{i\in I} \mathsf{y}_i\ \mathsf{d}_i.\mathsf{T}_i \quad \Big| \quad \mathsf{T}_1;\mathsf{T}_2 \quad \Big| \quad \mathsf{T}^\star \quad \Big| \quad \mathsf{end}$$

An LTT is either an internal ($\bigoplus$) or external ($\sum$) guarded choice on non-empty index sets I, the sequential composition $\_;\_$, an iteration $\_^\star$, or the empty term end. We usually omit trailing occurrences of end and write $\overline{\mathsf{y}_n}\ \mathsf{d}_n;\mathsf{T}_n$ (resp. $\mathsf{y}_n\ \mathsf{d}_n;\mathsf{T}_n$) for $\bigoplus_{i\in I} \overline{\mathsf{y}_i}\ \mathsf{d}_i.\mathsf{T}_i$ (resp. $\sum_{i\in I} \mathsf{y}_i\ \mathsf{d}_i.\mathsf{T}_i$) when $I = \{n\}$.

We assume that all channels appearing in the guards of an internal or an external choice are pairwise different. Similarly to GTT, the set $\mathtt{ch}(\mathsf{T})$ of *channels* of an LTT T is defined as

$$\mathtt{ch}(\bigoplus_{i\in I} \overline{\mathsf{y}_i}\ \mathsf{d}_i.\mathsf{T}_i) \quad = \quad \mathtt{ch}(\sum_{i\in I} \mathsf{y}_i\ \mathsf{d}_i.\mathsf{T}_i) \quad = \quad \{\mathsf{y}_i \mid i \in I\} \cup \bigcup_{i\in I} \mathtt{ch}(\mathsf{T}_i)$$

$$\mathtt{ch}(\mathsf{T}_1;\mathsf{T}_2) = \mathtt{ch}(\mathsf{T}_1) \cup \mathtt{ch}(\mathsf{T}_2) \qquad \mathtt{ch}(\mathsf{T}^\star) = \mathtt{ch}(\mathsf{T}) \qquad \mathtt{ch}(\mathsf{end}) = \emptyset$$

A *local type* is defined by an equation

$$\mathcal{T}(\vec{\mathsf{y}}) \triangleq \mathsf{T} \qquad \text{where} \qquad \mathtt{ch}(\mathsf{G}) \subseteq \vec{\mathsf{y}} \subseteq \mathbb{Y} \quad \text{and} \quad \vec{\mathsf{y}} \text{ are pairwise distinct names}$$

We abbreviate $\mathcal{T}(\vec{\mathsf{y}}) \triangleq \mathsf{T}$ with $\mathcal{T}(\vec{\mathsf{y}})$ when T is immaterial and we write $\mathcal{T}$ or T instead of $\mathcal{T}(\vec{\mathsf{y}})$ when parameters are understood.

The structural congruence on LTTs is defined as the smallest congruence $\equiv_{\mathrm{LTT}}$ such that

- internal and external choice operators are associative, commutative and have end as identity,
- and $\_;\_$ is associative and has end as neutral element.

Two local types $\mathcal{T}_1(\vec{\mathsf{y}}_1) \triangleq \mathsf{T}_1$ and $\mathcal{T}_2(\vec{\mathsf{y}}_2) \triangleq \mathsf{T}_2$ are structurally equivalent (written $\mathcal{T}_1 \equiv \mathcal{T}_2$) when $\mathsf{T}_1 \equiv_{\mathrm{LTT}} \mathsf{T}_2\{\vec{\mathsf{y}}_1/\vec{\mathsf{y}}_2\}$. In the following, we consider types up-to structural congruence.

The *projection* operation extracts local types from a global type; we restrict such operation on well-formed global types. Given a participant $\mathsf{r} \in \mathbb{P}$, the *projection* of a well-formed GTT G on r, denoted as $\mathsf{G}\upharpoonright\mathsf{r}$, is defined as follows:

$$\mathsf{G}\upharpoonright\mathsf{r} = \begin{cases} \mathsf{end} & \text{if } \mathsf{r} \notin \mathcal{P}(\mathsf{G}) \\ \bigoplus_{i\in I} \overline{\mathsf{y}_i}\ \mathsf{d}_i.\mathsf{G}_i\upharpoonright\mathsf{r} & \text{if } \mathsf{G} = \sum_{i\in I} \mathsf{r} \twoheadrightarrow \mathsf{q}_i : \mathsf{y}_i\ \mathsf{d}_i; \mathsf{G}_i \\ \sum_{i\in I_\mathsf{r}} \mathsf{y}_i\ \mathsf{d}_i.\mathsf{G}_i\upharpoonright\mathsf{r} + \sum_{i\in I\setminus I_\mathsf{r}} \mathsf{G}_i\upharpoonright\mathsf{r} & \text{if } \mathsf{G} = \sum_{i\in I_\mathsf{r}} \mathsf{p} \twoheadrightarrow \mathsf{q}_i : \mathsf{y}_i\ \mathsf{d}_i; \mathsf{G}_i + \sum_{i\in I\setminus I_\mathsf{r}} \mathsf{p} \twoheadrightarrow \mathsf{q}_i : \mathsf{y}_i\ \mathsf{d}_i; \mathsf{G}_i \\ & \mathsf{r} \neq \mathsf{p},\ \text{and } I_\mathsf{r} = \{i \in I \mid \mathsf{q}_i = \mathsf{r}\} \\ \mathsf{G}_1\upharpoonright\mathsf{r}; \mathsf{G}_2\upharpoonright\mathsf{r} & \text{if } \mathsf{G} = \mathsf{G}_1; \mathsf{G}_2 \\ (\mathsf{G}_1\upharpoonright\mathsf{r})^\star; \overline{b_1}\ \mathsf{d}_1; \ldots; \overline{b_n}\ \mathsf{d}_n & \text{if } \mathsf{G} = \mathsf{G}_1^{*^f},\ \bigcup\{f(\mathsf{p}) \mid \mathsf{p} \in \mathtt{ch}(f)\} = \{b_1\ \mathsf{d}_1, \ldots, b_n\ \mathsf{d}_n\}, \\ & \text{and } \mathsf{r} \in \mathtt{rdy}(\mathsf{G}_1) \\ (\mathsf{G}_1\upharpoonright\mathsf{r})^\star; b\ \mathsf{d} & \text{if } \mathsf{G} = \mathsf{G}_1^{*^f},\ f(\mathsf{r}) = b\ \mathsf{d} \end{cases}$$

When projecting a global type G over a participant name r that does not appear in it, produces the idle local type end. The remaining cases implicitly assume that $\mathsf{r} \in \mathcal{P}(\mathsf{G})$.

The projection of the (unique) selector of a branch results in the internal choice on the session channels. Dually, the projection on a receiver in a branch results in an external choice; observe that branches where the receiver is not r (i.e., $i \in I \setminus I_\mathsf{r}$) are treated differently from those where the receiver is r (i.e., $i \in I_\mathsf{r}$). We remark that the projected local type is well-defined when G is well-formed: the condition about knowledge of choices ensures that $\mathsf{r} \in \mathcal{P}(\mathsf{G}_i)$ for all $i \in I \setminus I_\mathsf{r}$; moreover, r is the receiver in the first interaction of each branch G_i.

The projection of a sequential composition is self-explanatory. Iterative GTTs are projected depending on whether the role is a controller or not. For $\mathsf{G} = \mathsf{G}_1^{*^f}$, recall that (by well-formedness) there is an iteration-controller $\mathsf{r} \in \mathtt{rdy}(\mathsf{G}_1)$; the projection of G on the controller generates an iterative local type, which corresponds to the projection of G_1, followed by the messages that signal the termination of the iteration to the remaining participants

(i.e., the messages $\overline{b_1}\ \mathsf{d}_1; \ldots; \overline{b_n}\ \mathsf{d}_n$). Dually, the projection on the other participants waits for the signal to exit the iteration (i.e., $b\ \mathsf{d}$). The *projection* $\mathcal{G}(\vec{\mathsf{y}})\upharpoonright \mathsf{r}$ of a global type $\mathcal{G}(\vec{\mathsf{y}}) \triangleq \mathsf{G}$ with respect to r is a local type $\mathcal{T}(\vec{\mathsf{y}}) \triangleq \mathsf{T}$ where $\mathsf{T} = \mathsf{G}\upharpoonright \mathsf{r}$.

Example 2.4. Consider the GTT G introduced in Example 2.1. Since G consists of a single branch where c is the selector, the projection on c is an internal choice with a single branch that sends a message on channel login and follows with the projection of the iterative type in the continuation. Since c is the interation-controller of the continuation, its projection consists of the iteration of the body followed by the termination message quit, as shown below.

$$\mathsf{G}\upharpoonright \mathsf{c} = \overline{\mathsf{login}}\ \mathsf{Str}.(\overline{\mathsf{deposit}}\ \mathsf{Int} + \overline{\mathsf{overdraft}}\ \mathsf{Int}.(\mathsf{ok} + \mathsf{ko}))^\star; \overline{\mathsf{quit}}$$

$$\mathsf{G}\upharpoonright \mathsf{b} = \mathsf{login}\ \mathsf{Str}.(\mathsf{deposit}\ \mathsf{Int} + \mathsf{overdraft}\ \mathsf{Int}.(\overline{\mathsf{ok}} + \overline{\mathsf{ko}}))^\star; \mathsf{quit}$$

Note that $\mathsf{G}\upharpoonright \mathsf{b}$ and $\mathsf{G}\upharpoonright \mathsf{b}$ are each other's dual. ◇

3. Types for the POP2 Protocol

We illustrate our approach on the Post Office Protocol Version 2 (POP2) [6] between a client c and a mail server s. We describe POP2 with the following global type:

$$\mathsf{G}_{\text{POP}} \triangleq \mathsf{c} \to \mathsf{s}: \mathsf{quit}\ ; \mathsf{G}_{\text{EXIT}} + \mathsf{c} \to \mathsf{s}: \mathsf{helo}\ \mathsf{Str}; \mathsf{G}_{\text{MBOX}}$$

$$\mathsf{G}_{\text{EXIT}} \triangleq \mathsf{s} \to \mathsf{c}: \mathsf{bye}$$

The global type G_{POP} starts with c sending a message to s either on channel quit (of the omitted sort Unit) or on helo to communicate its password (of sort Str). In the first case, the interaction ends after s sends a message on bye as per G_{EXIT}; in the latter case the protocol follows as per G_{MBOX} below.

$$\mathsf{G}_{\text{MBOX}} \triangleq \mathsf{s} \to \mathsf{c}: \mathsf{e}\ ; \mathsf{G}_{\text{EXIT}} + \mathsf{s} \to \mathsf{c}: \mathsf{r}\ \mathsf{Int}; \mathsf{G}_{\text{NMBR}}$$

In G_{MBOX}, s either signals an error on e before terminating or sends a message on r containing the number of messages in the default mailbox and then continues as G_{NMBR}:

$$\mathsf{G}_{\text{NMBR}} \triangleq (\ \mathsf{c} \to \mathsf{s}: \mathsf{fold}\ \mathsf{Str}; \mathsf{s} \to \mathsf{c}: \mathsf{r}\ \mathsf{Int} + \mathsf{c} \to \mathsf{s}: \mathsf{read}\ \mathsf{Int}; \mathsf{s} \to \mathsf{c}: \mathsf{r}\ \mathsf{Int}; \mathsf{G}_{\text{size}})^{*\mathsf{s}\mapsto\mathsf{quit}}; \mathsf{G}_{\text{EXIT}}$$

where c repeatedly requests either (a) the number of messages available in a folder, or (b) the length of a particular message in the current folder. The iteration-controller is c and it uses quit to communicate the termination of the loop to s. In case (a), c sends the folder's name over the channel fold and waits for the answer on r; after this, the body of the iteration is completed and the loop can be repeated again. In case (b), c sends the index corresponding to the selected message on channel read and waits for the answer on channel r; after this, the interaction continues as G_{size} specified below:

$$\mathsf{G}_{\text{size}} \triangleq (\ \mathsf{c} \to \mathsf{s}: \mathsf{read}\ \mathsf{Int}; \mathsf{s} \to \mathsf{c}: \mathsf{r}\ \mathsf{Int} + \mathsf{c} \to \mathsf{s}: \mathsf{retr}\ ; \mathsf{s} \to \mathsf{c}: \mathsf{msg}\ \mathsf{Data}.\mathsf{G}_{\text{xfer}})^{*\mathsf{s}\mapsto\mathsf{fold}\ \mathsf{Str}}; \mathsf{s} \to \mathsf{c}: \mathsf{r}\ \mathsf{Int}$$

In G_{size}, another loop controlled by c lets the client either (a) ask for another message by interacting again on read as described above or (b) retrieve a message. In the latter case, c signals on retr that it is ready to receive the message, which is then sent back on msg by s

(sort Data abstracts away the format of messages specified in [17]). Finally, c acknowledges the reception of the requested message as follows:

$$\begin{array}{rcl} \mathsf{G}_{\mathsf{xfer}} & \triangleq & \mathsf{c} \twoheadrightarrow \mathsf{s}\colon \mathsf{acks}\,;\mathsf{s} \twoheadrightarrow \mathsf{c}\colon \mathsf{r}\ \mathsf{Int} \\ & + & \mathsf{c} \twoheadrightarrow \mathsf{s}\colon \mathsf{ackd}\,;\mathsf{s} \twoheadrightarrow \mathsf{c}\colon \mathsf{r}\ \mathsf{Int} \\ & + & \mathsf{c} \twoheadrightarrow \mathsf{s}\colon \mathsf{nack}\,;\mathsf{s} \twoheadrightarrow \mathsf{c}\colon \mathsf{r}\ \mathsf{Int} \end{array}$$

Basically c may use one among three alternative channels: acks to acknowledge the reception of the message, ackd to keep the message or, nack to notify that the message has not been received properly (in which case the message is kept in the mailbox). In each case, s sends back to c the length of the next message over channel r.

The local type T_s obtained by projecting $\mathsf{G}_{\mathsf{POP}}$ on the participant s, i.e., $\mathsf{T}_\mathsf{s} = \mathsf{G}_{\mathsf{POP}}\upharpoonright\mathsf{s}$, is below.

$$\begin{array}{rcl} \mathsf{T}_\mathsf{s} & \triangleq & \mathsf{quit}\,.\mathsf{T}_{\mathsf{EXIT}} \ + \ \mathsf{helo}\ \mathsf{Str}.\mathsf{T}_{\mathsf{MBOX}} \\ \mathsf{T}_{\mathsf{EXIT}} & \triangleq & \overline{\mathsf{bye}} \\ \mathsf{T}_{\mathsf{MBOX}} & \triangleq & \overline{\mathsf{e}}\,.\mathsf{T}_{\mathsf{EXIT}} \ \oplus \ \overline{\mathsf{r}}\ \mathsf{Int}.\mathsf{T}_{\mathsf{NMBR}} \\ \mathsf{T}_{\mathsf{NMBR}} & \triangleq & (\mathsf{fold}\ \mathsf{Str}.\overline{\mathsf{r}}\ \mathsf{Int} \ + \ \mathsf{read}\ \mathsf{Int}.\overline{\mathsf{r}}\ \mathsf{Int}.\mathsf{T}_{\mathsf{size}})^*;\mathsf{quit}\,;\mathsf{T}_{\mathsf{EXIT}} \\ \mathsf{T}_{\mathsf{size}} & \triangleq & (\mathsf{read}\ \mathsf{Int}.\overline{\mathsf{r}}\ \mathsf{Int} \ + \ \mathsf{retr}\,.\overline{\mathsf{msg}}\ \mathsf{Data}.\mathsf{T}_{\mathsf{xfer}})^*;\mathsf{fold}\ \mathsf{Str};\overline{\mathsf{r}}\ \mathsf{int} \\ \mathsf{T}_{\mathsf{xfer}} & \triangleq & \mathsf{acks}\,.\overline{\mathsf{r}}\ \mathsf{Int} \ + \ \mathsf{ackd}\,.\overline{\mathsf{r}}\ \mathsf{Int} \ + \ \mathsf{nack}\,.\overline{\mathsf{r}}\ \mathsf{Int} \end{array}$$

Note that the messages in T_s are as in $\mathsf{G}_{\mathsf{POP}}$. We remark that s does not control any of the two iterations (i.e., $\mathsf{G}_{\mathsf{NMBR}}$ and $\mathsf{G}_{\mathsf{size}}$), hence the projections iterate until s receive a signal on the termination channels: quit in $\mathsf{T}_{\mathsf{NMBR}}$ and fold in $\mathsf{T}_{\mathsf{size}}$, respectively.

The projection $\mathsf{G}_{\mathsf{POP}}\upharpoonright\mathsf{c}$ of $\mathsf{G}_{\mathsf{POP}}$ on c is obtained analogously; the resulting local type is the dual of T_s, i.e., the one obtained by substituting internal choices by external ones and vice versa.

The projection $\mathsf{G}_{\mathsf{POP}}\upharpoonright\mathsf{c}$ of $\mathsf{G}_{\mathsf{POP}}$ onto c is obtained analogously; the resulting local type is the dual of T_s, i.e., the one obtained by substituting internal choices by external ones and vice versa.

For illustrative purpose, in the next example we present a multiparty variant of $\mathsf{G}_{\mathsf{POP}}$, where the authentication is outsourced.

Example 3.1. A multiparty variant of POP2 is defined by the global type $\mathsf{G}'_{\mathsf{POP}}$ below.

$$\begin{array}{rcl} \mathsf{G}'_{\mathsf{POP}} & \triangleq & \mathsf{c} \twoheadrightarrow \mathsf{s}\colon \mathsf{quit}\,;\mathsf{G}_{\mathsf{EXIT}} \ + \ \mathsf{c} \twoheadrightarrow \mathsf{s}\colon \mathsf{helo}\ \mathsf{Str};\mathsf{G}'_{\mathsf{MBOX}} \\ \mathsf{G}'_{\mathsf{MBOX}} & \triangleq & \mathsf{s} \twoheadrightarrow \mathsf{a}\colon \mathsf{req}\ \mathsf{Str};\mathsf{a} \twoheadrightarrow \mathsf{s}\colon \mathsf{res}\ \mathsf{Bool};\big(\mathsf{s} \twoheadrightarrow \mathsf{c}\colon \mathsf{e}\,;\mathsf{G}_{\mathsf{EXIT}} \ + \ \mathsf{s} \twoheadrightarrow \mathsf{c}\colon \mathsf{r}\ \mathsf{Int};\mathsf{G}_{\mathsf{NMBR}}\big) \end{array}$$

In this version, s uses a third-party authentication service a, which is contacted immediately after the server receives a helo message from the client. The server sends to a an authentication request over res and waits for the authorisation on res ($\mathsf{G}_{\mathsf{NMBR}}$ and $\mathsf{G}_{\mathsf{EXIT}}$ remain unchanged).

The following equations

$$\begin{array}{rcl} \mathsf{T}'_\mathsf{s} & \triangleq & \mathsf{quit}\,.\mathsf{T}_{\mathsf{EXIT}} \ + \ \mathsf{helo}\ \mathsf{Str}.\mathsf{T}_{\mathsf{AUTH}} \\ \mathsf{T}_{\mathsf{AUTH}} & \triangleq & \overline{\mathsf{req}}\ \mathsf{Str}.\mathsf{res}\ \mathsf{Bool}.\mathsf{T}'_{\mathsf{MBOX}} \\ \mathsf{T}'_{\mathsf{MBOX}} & \triangleq & \overline{\mathsf{e}}\,.\mathsf{T}_{\mathsf{EXIT}} \ \oplus \ \overline{\mathsf{r}}\ \mathsf{Int}.\mathsf{T}_{\mathsf{NMBR}} \end{array}$$

yield the projection $\mathsf{T}'_\mathsf{s}\mathsf{G}'_{\mathsf{POP}}\upharpoonright\mathsf{s}$ of $\mathsf{G}'_{\mathsf{POP}}$ on s. ◇

P, Q	::=	**processes**	N	::=	**input-guarded processes**
	$\overline{u}^{\mathsf{n}}(\vec{\mathsf{y}}).P$	request		$\sum_{i\in I}\mathsf{y}_i(x_i).P_i$	choices
	$u^{\mathsf{p}}(\vec{\mathsf{y}}).P$	accept			
	N	choice			
	$\overline{\mathsf{y}}\, e$	send	S	::=	**systems**
	$P;Q$	sequential		P	
	if e then P else Q	conditional		$S\|S$	parallel
	for x in ℓ do P	for		$\mathsf{y}[\vec{\mathsf{v}}]$	queue
	repeat N until N	repeat		$(\nu\vec{\mathsf{y}}@u)S$	restriction

Figure 2: Syntax of processes and systems.

4. Processes and Systems

Choreographies specify distributed applications that we refer to as *systems*. Concretely, systems are the parallel composition of *processes* realising the roles in a choreography.

Processes manipulate and exchange values obtained by evaluating expressions. Let $\mathbb{X}$ and $\mathbb{V}$ be two infinite disjoint sets of *variables* and *basic values* respectively which are both disjoint from the sets of shared names $\mathbb{U}$, session channels $\mathbb{Y}$, and participants $\mathbb{P}$. Values are specified by *expressions* having the following syntax:

$$e \;::=\; x \;\mid\; \mathsf{v} \;\mid\; e_1\ \mathtt{bop}\ e_2 \;\mid\; \mathtt{uop}\ e \qquad\qquad \ell \;::=\; [e_1,\ldots,e_n] \;\mid\; e_1..e_2$$

An expression e is either a variable $x \in \mathbb{X}$ or a value $\mathsf{v} \in \mathbb{V}$, or else the composition $e_1\ \mathtt{bop}\ e_2$ of two expressions through a binary operator $\mathtt{bop}$, or the application of a unary operator $\mathtt{uop}$ to an expression (operators are left unspecified and can be thought of as the usual logical-arithmetic operators of programming languages). We assume that expressions are implicitly sorted and, for simplicity, our expressions do not include binders of variables, names, or test for definiteness. Lists $[e_1,\ldots,e_n]$ and numerical ranges $e_1..e_2$ are used for iteration; in the latter case, both expressions e_1 and e_2 are of sort integer. The empty list is denoted as ε and the operations $\mathsf{hd}(\ell)$ and $\mathsf{tl}(\ell)$ respectively return the head and tail of ℓ (defined as usual). Given an expression e or a list ℓ, the sets $\mathrm{var}(e)$ and $\mathrm{var}(\ell)$ of *variables of e and ℓ* respectively, are defined as

$$\mathrm{var}(x)=\{x\} \quad \mathrm{var}(\mathsf{v})=\emptyset \quad \mathrm{var}(\mathtt{uop}\ e)=\mathrm{var}(e) \quad \mathrm{var}(e_1\ \mathtt{bop}\ e_2)=\mathrm{var}(e_1)\cup\mathrm{var}(e_2)$$

$$\mathrm{var}([e_1,\ldots,e_n])=\textstyle\bigcup_{i=1}^{n}\mathrm{var}(e_i) \qquad \mathrm{var}(e_1..e_2)=\mathrm{var}(e_1)\cup\mathrm{var}(e_2)$$

The syntax of processes P, input-guarded non-deterministic sequential processes N, and systems S is given in Figure 2. A process $\overline{u}^{\mathsf{n}}(\vec{\mathsf{y}}).P$ requests a new session on a shared name u and then behaves as P; dually, process $u^{\mathsf{p}}(\vec{\mathsf{y}}).P$ accepts the request of a new session from another process and then behaves as P. A message e is sent on a session channel y by the process $\overline{\mathsf{y}}\, e$. Sequential composition and conditional are standard. An input-guarded non-deterministic sequential processes $\sum_{i\in I}\mathsf{y}_i(x_i).P_i$ (conventionally denoted as $\mathsf{0}$ when $I=\emptyset$) can branch over P_i when a message is received on the session channel y_i; we assume $\mathsf{y}_i \neq \mathsf{y}_j$ when $i \neq j \in I$. Our language for processes provide two different constructs for iterations

$$\text{for } x \text{ in } \ell \text{ do } P \qquad \text{and} \qquad \text{repeat } N \text{ until } \sum_{i\in I}\mathsf{y}_i(x_i).P_i$$

Intuitively, the former realises the controllers of iterative global types, while the latter is used for the remaining roles (*cf.* Section 7). A for-loop iterates the body P on the list ℓ. The body N in a repeat-until-loop is a process of the form $\sum_{i\in I} \mathsf{y}_i(x_i).P_i$ and it is repeated until a message on one of the channels y_i on the until guard is received.

We set the following precedence rules: if _ then _ else _, for _ in _ do _ and repeat _ until have the lowest precedence while _.._ has precedence over _;_ so that, e.g., the term if e then P else $u^{\mathsf{i}}(\vec{\mathsf{y}}).Q$ reads if e then P else $(u^{\mathsf{i}}(\vec{\mathsf{y}}).Q)$ and $u^{\mathsf{i}}(\vec{\mathsf{y}}).P;Q$ reads $(u^{\mathsf{i}}(\vec{\mathsf{y}}).P);Q$.

Systems consist of a parallel composition of process together with the queues $\mathsf{y}[\vec{\mathsf{v}}]$ that store the values $\vec{\mathsf{v}}$ sent over the session channels y. Given $\mathsf{y}_1,\dots,\mathsf{y}_h$ pairwise distinct session channels, we write $(\mathsf{y}_1,\dots,\mathsf{y}_h)[\vec{\mathsf{v}}_1,\dots,\vec{\mathsf{v}}_h]$ to denote $\mathsf{y}_1[\vec{\mathsf{v}}_1] \mid \dots \mid \mathsf{y}_h[\vec{\mathsf{v}}_h]$. Names $\vec{\mathsf{y}}$ are bound in $(\nu\vec{\mathsf{y}}@u)S$ and related to the shared name u.

The definition of the set $\mathtt{fn}(\_)$ of *free names* is standard but for shared names u which are also decorated to keep track of roles; formally we define $\mathtt{fn}(\_)$ on systems as

$$\mathtt{fn}(\mathsf{y}[\vec{\mathsf{v}}]) = \{\mathsf{y}\} \qquad \mathtt{fn}((\nu\vec{\mathsf{y}}@u)S) = \mathtt{fn}(S) \setminus \vec{\mathsf{y}} \qquad \mathtt{fn}(S|S') = \mathtt{fn}(S) \cup \mathtt{fn}(S')$$

while for processes we have

$$\begin{array}{ll}
\mathtt{fn}(\overline{u}^{\mathsf{n}}(\vec{\mathsf{y}}).P) = \{u, u^0\} \cup \mathtt{fn}(P) \setminus \vec{\mathsf{y}} & \mathtt{fn}(u^{\mathsf{p}}(\vec{\mathsf{y}}).P) = \{u, u^{\mathsf{p}}\} \cup \mathtt{fn}(P) \setminus \vec{\mathsf{y}} \\
\mathtt{fn}(\overline{\mathsf{y}}\, e) = \{\mathsf{y}\} \cup \mathrm{var}(e) & \mathtt{fn}(\sum_{i\in I} \mathsf{y}_i(x_i).P_i) = \bigcup_{i\in I}(\{\mathsf{y}_i\} \cup \mathtt{fn}(P_i) \setminus \{x_i\}) \\
\mathtt{fn}(P;Q) = \mathtt{fn}(P) \cup \mathtt{fn}(Q) & \mathtt{fn}(\text{if } e \text{ then } P \text{ else } Q) = \mathtt{fn}(P) \cup \mathtt{fn}(Q) \cup \mathrm{var}(e) \\
\mathtt{fn}(\text{for } x \text{ in } \ell \text{ do } P) = \mathtt{fn}(P) \setminus \{x\} \cup \mathtt{fn}(\ell) & \mathtt{fn}(\text{repeat } N \text{ until } N') = \mathtt{fn}(N) \cup \mathtt{fn}(N')
\end{array}$$

where, in the first equation, a process requesting a new session on u plays the role 0 and, in the second equation, a process accepting on u^{p} plays role p.

The set $\mathtt{fu}(S)$ of *free shared names* of S is defined as $\mathtt{fn}(S) \cap \mathbb{U}$. Similarly the set of *free session names* $\mathtt{fy}(S)$ (resp. *free variables* $\mathtt{fx}(S)$) of S is defined as $\mathtt{fn}(S) \cap \mathbb{Y}$ (resp. $\mathtt{fn}(S) \cap \mathbb{X}$). The set $\mathtt{bn}(\_)$ of *bound names* is defined as

$$\begin{array}{ll}
\mathtt{bn}(\overline{u}^{\mathsf{n}}(\vec{\mathsf{y}}).P) = \vec{\mathsf{y}} \cup \mathtt{bn}(P) & \mathtt{bn}(u^{\mathsf{p}}(\vec{\mathsf{y}}).P) = \vec{\mathsf{y}} \cup \mathtt{bn}(P) \\
\mathtt{bn}(\overline{\mathsf{y}}\, e) = \emptyset & \mathtt{bn}(\sum_{i\in I} \mathsf{y}_i(x_i).P_i) = \bigcup_{i\in I}(\{x_i\} \cup \mathtt{bn}(P_i)) \\
\mathtt{bn}(P;Q) = \mathtt{bn}(P) \cup \mathtt{bn}(Q) & \mathtt{bn}(\text{if } e \text{ then } P \text{ else } Q) = \mathtt{bn}(P) \cup \mathtt{bn}(Q) \\
\mathtt{bn}(\text{for } x \text{ in } \ell \text{ do } P) = \{x\} \cup \mathtt{bn}(P) & \mathtt{bn}(\text{repeat } N \text{ until } N') = \mathtt{bn}(N) \cup \mathtt{bn}(N') \\
\mathtt{bn}(\mathsf{y}[\vec{\mathsf{v}}]) = \emptyset & \mathtt{bn}((\nu\vec{\mathsf{y}}@u)S) = \vec{\mathsf{y}} \cup \mathtt{bn}(S) \\
\mathtt{bn}(S|S') = \mathtt{bn}(S) \cup \mathtt{bn}(S') &
\end{array}$$

The set $\mathtt{by}(S)$ of *bound session names* of S is defined as $\mathtt{bn}(S) \cap \mathbb{Y}$ and the set $\mathtt{bx}(S)$ of *bound variables* of S is $\mathtt{bn}(S) \cap \mathbb{X}$. Note that $\mathtt{bn}(S) \cap \mathbb{U} = \emptyset$ for all S.

Figure 3 summarises the notation introduced so far for the syntax of processes.

As customary, we rely on a *structural congruence relation* $\equiv$ defined as the least congruence over systems closed with respect to α-conversion and the following axioms

$$\begin{array}{ll}
(\nu\vec{\mathsf{y}}@u)(\nu\vec{\mathsf{y}}'@u')S \equiv (\nu\vec{\mathsf{y}}'@u')(\nu\vec{\mathsf{y}}@u)S & (\nu\vec{\mathsf{y}}@u)0 \equiv 0 \\
(\nu\vec{\mathsf{y}}@u)(S|S') \equiv S \mid (\nu\vec{\mathsf{y}}@u)S', \text{ when } \vec{\mathsf{y}} \not\subseteq \mathtt{fy}(S) &
\end{array}$$

and such that _|_ and _;_ form monoids with identity 0 and the former is commutative.

The operational semantics of systems is given by the LTS inductively defined by the rules in Figures 4 and 5 where

Sets

Set	Elements	Description
$\mathbb{X}$	$x, x_1, \ldots$	variables
$\mathbb{V}$	$\mathsf{v}, \mathsf{v}_1, \ldots$	basic values
$\mathbb{U}$	$u, u_1, \ldots$	shared names
$\mathbb{Y}$	$\mathsf{y}, \mathsf{s}, \ldots$	session channels
$\mathbb{P}$	$\mathsf{p}, \mathsf{q}, \mathsf{r}, \ldots$	participants
e	$e, e_1, \ldots$	expressions
ℓ	$\ell, \ell_1, \ldots$	lists

Functions

Function	Description
$\mathrm{var}(e), \mathrm{var}(\ell)$	variables of a expression, a list
$\mathtt{fn}(P), \mathtt{fn}(S)$	free names of a process, a system
$\mathtt{bn}(P), \mathtt{fn}(S)$	bound names of a process, a system
$\mathtt{by}(P), \mathtt{by}(S)$	bound session names of a process, a system
$\mathtt{bx}(P), \mathtt{bx}(S)$	of bound variables of a process, a system

Figure 3: Summary of notation for processes

- a store σ records both the values assigned to variables and the session channels created by a process,
- $\sigma[x \mapsto \mathsf{v}]$ is the update of σ at x with v (and likewise for $\sigma[\mathsf{y} \mapsto u]$), and
- $e \downarrow \sigma$ is the evaluation of e (defined if $\mathrm{var}(e) \subseteq \mathrm{dom}(\sigma)$ and undefined otherwise). We assume that an expression e depends only on its variables, that is, for all stores σ and σ':

$$\sigma|_{\mathbb{X}} = \sigma'|_{\mathbb{X}} \implies e \downarrow \sigma = e \downarrow \sigma'$$

where $\cdot|_{\_}$ is the standard restriction of a function on a subset of its domain.

Labels are given by the following productions

$$\alpha \ ::= \ \overline{u}^{\mathsf{n}}(\vec{\mathsf{y}}) \quad | \quad u^{\mathsf{p}}(\vec{\mathsf{y}}) \quad | \quad \overline{\mathsf{y}}\mathsf{v} \quad | \quad \mathsf{y}\mathsf{v} \quad | \quad \tau \quad | \quad e \vdash \alpha \qquad (4.1)$$

that respectively represent the request of initialisation of a session on u, the acceptance of joining a session on u with role p, the sending of a value on y, the reception of a value on y, the silent step τ, and *conditional* actions $e \vdash \alpha$ where e is a boolean expression. A conditional action labelling a transition $\langle S, \sigma\rangle \xrightarrow{e \vdash \alpha} \langle S', \sigma'\rangle$ denotes that $\langle S, \sigma\rangle$ performs the action α and moves to $\langle S', \sigma'\rangle$ because $e \downarrow \sigma$ holds. We may write α instead of $\mathtt{true} \vdash \alpha$ and $e \wedge e' \vdash \alpha$ instead of $e \vdash (e' \vdash \alpha)$.

Functions $\mathtt{fy}(\_)$ and $\mathtt{bn}(\_)$ extend to labels as follows:

$$\mathtt{fy}(\overline{u}^{\mathsf{n}}(\vec{\mathsf{y}})) = \mathtt{fy}(u^{\mathsf{p}}(\vec{\mathsf{y}})) = \{u\} \quad \mathtt{fy}(\overline{\mathsf{y}}\mathsf{v}) = \mathtt{fy}(\mathsf{y}\mathsf{v}) = \{\mathsf{y}\} \quad \mathtt{fy}(\tau) = \emptyset \quad \mathtt{fy}(e \vdash \alpha) = \mathtt{fy}(\alpha)$$

$$\mathtt{bn}(\overline{u}^{\mathsf{p}}(\vec{\mathsf{y}})) = \mathtt{bn}(u^{\mathsf{p}}(\vec{\mathsf{y}})) = \vec{\mathsf{y}} \qquad \mathtt{bn}(\overline{\mathsf{y}}\mathsf{v}) = \mathtt{bn}(\mathsf{y}\mathsf{v}) = \mathtt{bn}(\tau) = \emptyset \qquad \mathtt{bn}(e \vdash \alpha) = \mathtt{bn}(\alpha)$$

We comment on the rules in Figure 4 and 5. In Figure 4, rules [SReq] and [SAcc] deal with the initialisation of new sessions; the store is updated to keep track of the fresh session channels $\vec{\mathsf{y}}$ used in the choreography u (implicitly α-converting $\vec{\mathsf{y}}$ when $\vec{\mathsf{y}} \in \mathrm{dom}(\sigma)$). Rule [SSend] is for sending values. Rule [SRcv] is for receiving messages in an early style approach

$$[\mathsf{SReq}]\ \frac{\vec{\mathsf{y}} \notin \mathrm{dom}(\sigma)}{\langle \overline{u}^{\mathsf{n}}(\vec{\mathsf{y}}).P, \sigma\rangle \xrightarrow{\overline{u}^{\mathsf{n}}(\vec{\mathsf{y}})} \langle P, \sigma[\vec{\mathsf{y}} \mapsto u]\rangle} \qquad [\mathsf{SAcc}]\ \frac{\vec{\mathsf{y}} \notin \mathrm{dom}(\sigma)}{\langle u^{\mathsf{p}}(\vec{\mathsf{y}}).P, \sigma\rangle \xrightarrow{u^{\mathsf{p}}(\vec{\mathsf{y}})} \langle P, \sigma[\vec{\mathsf{y}} \mapsto u]\rangle}$$

$$[\mathsf{SSend}]\ \frac{e \downarrow \sigma = \mathsf{v}}{\langle \overline{\mathsf{y}}\, e, \sigma\rangle \xrightarrow{\overline{\mathsf{y}}\mathsf{v}} \langle 0, \sigma\rangle}$$

$$[\mathsf{SRcv}]\ \langle \sum_{i\in I} \mathsf{y}_i(x_i).P_i, \sigma\rangle \xrightarrow{\mathsf{y}_j\mathsf{v}} \langle P_j, \sigma[x_j \mapsto \mathsf{v}]\rangle\ j \in I$$

$$[\mathsf{SSeq}]\ \frac{\langle P, \sigma\rangle \xrightarrow{e\vdash\alpha} \langle P', \sigma'\rangle}{\langle P; Q, \sigma\rangle \xrightarrow{e\vdash\alpha} \langle P'; Q, \sigma'\rangle}$$

$$[\mathsf{SThen}]\ \frac{e \downarrow \sigma = \mathtt{true} \quad \langle P, \sigma\rangle \xrightarrow{e'\vdash\alpha} \langle P', \sigma'\rangle}{\langle \mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q, \sigma\rangle \xrightarrow{e\wedge e'\vdash\alpha} \langle P', \sigma'\rangle} \qquad [\mathsf{SElse}]\ \frac{e \downarrow \sigma = \mathtt{false} \quad \langle Q, \sigma\rangle \xrightarrow{e'\vdash\alpha} \langle Q', \sigma'\rangle}{\langle \mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q, \sigma\rangle \xrightarrow{\neg e\wedge e'\vdash\alpha} \langle Q', \sigma'\rangle}$$

$$[\mathsf{SFor}]\ \frac{\ell \downarrow \sigma \neq \varepsilon \qquad \langle P, \sigma[x \mapsto \mathsf{hd}(\ell \downarrow \sigma)]\rangle \xrightarrow{e\vdash\alpha} \langle P', \sigma'\rangle}{\langle \mathsf{for}\ x\ \mathsf{in}\ \ell\ \mathsf{do}\ P, \sigma\rangle \xrightarrow{e\vdash\alpha} \langle P'; \mathsf{for}\ x\ \mathsf{in}\ \mathsf{tl}(\ell)\ \mathsf{do}\ P, \sigma'\rangle} \qquad [\mathsf{SForEnd}]\ \frac{\ell \downarrow \sigma = \varepsilon}{\langle \mathsf{for}\ x\ \mathsf{in}\ \ell\ \mathsf{do}\ P, \sigma\rangle \xrightarrow{\tau} \langle 0, \sigma\rangle}$$

$$[\mathsf{SLoop}]\ \frac{\langle N, \sigma\rangle \xrightarrow{e\vdash\alpha} \langle P, \sigma'\rangle \quad M = \sum_{i\in I} \mathsf{y}_i(x_i).P_i \quad \forall i \in I.\mathsf{y}_i \notin \mathtt{fy}(\alpha)}{\langle \mathsf{repeat}\ N\ \mathsf{until}\ M, \sigma\rangle \xrightarrow{e\vdash\alpha} \langle P; \mathsf{repeat}\ N\ \mathsf{until}\ M, \sigma'\rangle} \qquad [\mathsf{SLoopEnd}]\ \frac{\langle M, \sigma\rangle \xrightarrow{e\vdash\alpha} \langle P, \sigma'\rangle}{\langle \mathsf{repeat}\ N\ \mathsf{until}\ M, \sigma\rangle \xrightarrow{e\vdash\alpha} \langle P, \sigma'\rangle}$$

Figure 4: Labelled transitions for processes

$$[\mathsf{SInit}]\ \langle \overline{u}^{\mathsf{n}}(\vec{\mathsf{y}}).P_0 | u^1(\vec{\mathsf{y}}).P_1 | \ldots | u^{\mathsf{n}}(\vec{\mathsf{y}}).P_n, \sigma\rangle \xrightarrow{\tau} \langle (\nu\vec{\mathsf{y}}@u)(P_0 | \ldots | P_n | \vec{\mathsf{y}} : \emptyset), \sigma[\vec{\mathsf{y}} \mapsto u]\rangle \qquad \vec{\mathsf{y}} \notin \mathrm{dom}(\sigma)$$

$$[\mathsf{SCom}_1]\ \frac{\langle P, \sigma\rangle \xrightarrow{e\vdash\overline{\mathsf{y}}\mathsf{v}} \langle P', \sigma'\rangle}{\langle P \mid \mathsf{y}[\vec{\mathsf{v}}], \sigma\rangle \xrightarrow{e\vdash\tau} \langle P' \mid \mathsf{y}[\vec{\mathsf{v}} \cdot \mathsf{v}], \sigma'\rangle} \qquad [\mathsf{SCom}_2]\ \frac{\langle P, \sigma\rangle \xrightarrow{e\vdash\mathsf{y}\mathsf{v}} \langle P', \sigma'\rangle}{\langle P \mid \mathsf{y}[\mathsf{v} \cdot \vec{\mathsf{v}}], \sigma\rangle \xrightarrow{e\vdash\tau} \langle P' \mid \mathsf{y}[\vec{\mathsf{v}}], \sigma'\rangle}$$

$$[\mathsf{SPar}]\ \frac{\langle S_1, \sigma\rangle \xrightarrow{e\vdash\alpha} \langle S_1', \sigma'\rangle \qquad \mathtt{bn}(\alpha) \cap \mathrm{dom}(\sigma) = \emptyset \qquad \mathtt{fx}(S_2) \cap (\mathrm{dom}(\sigma') \setminus \mathrm{dom}(\sigma)) = \emptyset}{\langle S_1 | S_2, \sigma\rangle \xrightarrow{e\vdash\alpha} \langle S_1' | S_2, \sigma'\rangle}$$

$$[\mathsf{SNew}]\ \frac{\langle S, \sigma\rangle \xrightarrow{e\vdash\alpha} \langle S', \sigma'\rangle \qquad \vec{\mathsf{y}} \cap \mathtt{fy}(\alpha) = \emptyset}{\langle (\nu\vec{\mathsf{y}}@u)S, \sigma\rangle \xrightarrow{e\vdash\alpha} \langle (\nu\vec{\mathsf{y}}@u)S', \sigma'\rangle} \qquad [\mathsf{SStr}]\ \frac{S \equiv S_1 \qquad \langle S_1, \sigma\rangle \xrightarrow{e\vdash\alpha} \langle S_2, \sigma'\rangle \qquad S_2 \equiv S'}{\langle S, \sigma\rangle \xrightarrow{e\vdash\alpha} \langle S', \sigma'\rangle}$$

Figure 5: Labelled transitions for systems

(variables are assigned when firing an input prefix); the store is updated by recording that v is is assigned to x. Rule [SSeq] is for sequential composition. Rules [SThen] and [SElse] handle conditional statements as expected; their only peculiarity is that the guard is recorded on the label of the transition, which is instrumental for establishing the correspondence between

systems and their types (*cf.* Section 8). Rules [SFor], [SForEnd], [SLoop], and [SLoopEnd] unfold the corresponding iterative process as expected.

We now comment on the rules in Figure 5. Rule [SInit] synchronises n roles with the process $\overline{u}^{\mathsf{n}}(\vec{y}_0).P_0$ initialising the session; this creates a new session with (initially empty) queues on fresh session names $\vec{y}$. These queues are used to exchange values as prescribed by rules [SCom_1] and [SCom_2]. Communication actions of processes become silent at system level capturing the fact that each action is performed over a session queue. Rule [SPar] stands for those transitions involving just some of the components in a system. By the condition $\mathtt{bn}(\alpha) \cap \mathrm{dom}(\sigma) = \emptyset$ in the premiss of [SPar], α should contain fresh session names when it corresponds to the creation of a new session (i.e., it is either $\overline{u}^{\mathsf{n}}(\vec{y})$ or $u^{\mathsf{p}}(\vec{y})$). The rightmost condition in the premiss of rule [SPar] ensures that each process has its own local (logical) store (i.e., there is no confusion between bound variables of different processes). Rule [SNew] is standard and allows an action α to be observed only if it does not involve restricted names. Rule [SStr] is standard.

5. Processes of the POP2 Protocol

We now present an implementation for the role s of the global type G_{POP} introduced in Section 3. To ease the presentation, we abstract away from the concrete representation of folders and use the following auxiliary abstract operations:

auth	: Str → Bool	the authentication predicate,
msgs	: Str → Int	maps a folder name into the number of messages in that folder,
len	: Int → Int	maps a message index into the length of the message,
data	: Int → Data	maps a message index into its content,
next	: Int → Int	maps a message index to the next index in the folder,
del	: Int → Int	maps a message index to the next index in the folder after deletion.

As specified for POP2 [6], the value inbox of sort Int denotes the default folder.

Process *Init* below gives an implementation of the role s in the protocol POP2 described in Section 3.

$$\mathit{Init} \triangleq u^{\mathsf{s}}(\vec{y}).\mathit{Srv} \tag{5.1}$$

Init starts by joining a session on the shared channel u; it plays role s over the session channels $\vec{y} = (\mathsf{quit}, \mathsf{helo}, \mathsf{bye}, \mathsf{r}, \mathsf{e}, \mathsf{fold}, \mathsf{read}, \mathsf{retr}, \mathsf{msg}, \mathsf{acks}, \mathsf{ackd}, \mathsf{nack})$. Once the session is initiated, the continuation *Srv* implements the local type T_{s} in Section 3.

$$\mathit{Srv} \triangleq \mathsf{quit}.\mathit{Exit} + \mathsf{helo}(c).\mathit{Mbox}(c) \qquad \text{where} \qquad \mathit{Exit} \triangleq \overline{\mathsf{bye}} \tag{5.2}$$

As specified by T_{s}, *Srv* waits for a message on either quit or helo. After receiving a message on quit, it sends a message on bye and terminates, as defined by *Exit*. If the client sends instead its credentials over channel helo, then the implementation follows with $\mathit{Mbox}(c)$:

$$\mathit{Mbox}(c) \triangleq \mathsf{if}\ (\mathsf{auth}\ c)\ \mathsf{then}\ \overline{\mathsf{r}}\,(\mathsf{msgs}\ \mathsf{inbox}); \mathit{Nmbr}\ \mathsf{else}\ (\overline{\mathsf{e}}\,; \mathit{exit}) \tag{5.3}$$

$\mathit{Mbox}(c)$ resolves the non-deterministic choice in $\mathsf{T}_{\text{MBOX}} = \overline{\mathsf{e}}\,.\mathsf{T}_{\text{EXIT}} \oplus \overline{\mathsf{r}}\ \mathsf{Int}.\mathsf{T}_{\text{NMBR}}$ with a conditional statement that evaluates the credentials provided by the client. When they are valid, the implementation sends the number of messages in the default folder inbox over r and proceeds as *Nmbr* below. On the contrary, the process closes the session after sending messages over e and bye.

Process Nmbr implements the iterative behaviour defined by the local type T_{NMBR}:

$$\mathit{Nmbr} \triangleq \mathsf{repeat}\ \mathsf{fold}(f).\overline{\mathsf{r}}\,(\mathsf{msgs}\ f)\ +\ \mathsf{read}(m).\overline{\mathsf{r}}\,(\mathsf{len}\ m); \mathit{Size}(m)\ \mathsf{until}\ \mathsf{quit}.\mathit{Exit} \tag{5.4}$$

Since s is not the iteration-controller, its implementation uses a repeat-until-loop in which a client can repeatedly ask for the length of a folder (by using fold) or retrieve messages (by using read) until it terminates the interaction by sending a message over quit.

Processes $\mathit{Size}(m)$ and $\mathit{Xfer}(m)$ are the implementations of the local types T_{Size} and T_{Xfer}:

$$\mathit{Size}(m) \triangleq \mathsf{repeat}\ \mathsf{retr}.\overline{\mathsf{msg}}\,(\mathsf{data}\ m); \mathit{Xfer}(m)\ +\ \mathsf{read}(m).\overline{\mathsf{r}}\,(\mathsf{len}\ m)\ \mathsf{until}\ \mathsf{fold}(f).\overline{\mathsf{r}}\,(\mathsf{msgs}\ f)$$

$$\mathit{Xfer}(m) \triangleq \mathsf{acks}.\overline{\mathsf{r}}\,(\mathsf{len}\ (\mathsf{next}\ m))\ +\ \mathsf{ackd}.\overline{\mathsf{r}}\,(\mathsf{len}\ (\mathsf{del}\ m))\ +\ \mathsf{nack}.\overline{\mathsf{r}}\,(\mathsf{len}\ m)$$

Let G'_{POP} be the multiparty variant of POP2 introduced in Example 3.1. The process Init' below is a possible implementation of s in G'_{POP} (i.e., T'_{s}).

$$\begin{aligned} \mathit{Init}' &\triangleq u^{\mathsf{s}}(\vec{\mathsf{y}}).\mathit{Srv}' \\ \mathit{Srv}' &\triangleq \mathsf{quit}.\mathit{Exit}\ +\ \mathsf{helo}(c).\mathit{Auth}(c) \\ \mathit{Auth}(c) &\triangleq \overline{\mathsf{req}}\,c; \mathsf{res}(a).\mathit{Mbox}' \\ \mathit{Mbox}'(c,a) &\triangleq \mathsf{if}\ (\mathsf{auth}\ c) \wedge a\ \mathsf{then}\ \overline{\mathsf{r}}\,(\mathsf{msgs}\ \mathsf{inbox}); \mathit{Nmbr}\ \mathsf{else}\ \overline{\mathsf{e}}\,.\mathit{Exit} \end{aligned} \tag{5.5}$$

Init' is analogous to Init in (5.1); we remark that $\vec{\mathsf{y}}$ now includes also the session channels req and res used for interacting with the authorisation authority. After receiving the credentials of the client on session channel helo, the server interacts with the authorisation authority as defined in $\mathit{Auth}(c)$: it forwards the credentials over session channel req and awaits for the authorisation outcome a on session channel res. Finally, $\mathit{Mbox}'(c,a)$ resolves the non-deterministic choice in T_{Mbox} by taking into account both the authorisation outcome a and the client's credentials c. In this variant, a client can access the inbox only if the credentials satisfy *both* the local authentication function *and* the external authentication service.

6. Whole-Spectrum Implementation

In this section we formally characterise the whole-spectrum implementations of a role in a global type. We start by introducing the notion of (candidate) implementation of a global type, that is, a system in which each role of the global type is implemented by a process. The following definition syntactically characterises the processes that can play a specific role p in the implementation of a global type, i.e., those processes that are able to open a session to play role p.

Definition 6.1 (Unique role)**.** A process P *uniquely plays role* p *in* u if either of the following cases holds

- $P = \overline{u}^{\mathsf{n}}(\vec{\mathsf{y}}).Q$, $u \notin \mathtt{fn}(Q)$, and $\mathsf{p} = 0$
- $P = u^{\mathsf{p}}(\vec{\mathsf{y}}).Q$ and $u \notin \mathtt{fn}(Q)$
- $P = \sum_{i \in I} \mathsf{y}_i(x_i).Q_i$ and Q_i uniquely plays role p in u for each $i \in I$
- $P = \mathsf{if}\ e\ \mathsf{then}\ Q_1\ \mathsf{else}\ Q_2$ and both Q_1 and Q_2 uniquely play role p in u
- $P = Q_1; Q_2$ and either Q_1 uniquely plays role p in u and $u \notin \mathtt{fn}(Q_2)$ or Q_2 uniquely plays role p in u and $u \notin \mathtt{fn}(Q_1)$

- $P = \mathsf{repeat}\ Q_1\ \mathsf{until}\ Q_2$ and Q_2 uniquely plays role $\mathtt{p}$ in u and $u \notin \mathtt{fn}(Q_1)$.

For technical simplicity, we require a process playing role $\mathtt{p}$ in u to open just one session over the shared channel u (note the restriction $u \notin \mathtt{fn}(Q)$ in the first two items of the definition); a process playing different roles in several instances of the same global type can be handled by using different shared names associated to the same global type. For branches and conditional forms we require the process to play role $\mathtt{p}$ in u regardless of the chosen branch (e.g., in every continuation Q_j of a branching process). The case for sequential composition is straightforward. We remark that $\overline{\mathsf{y}}\, e$ does not play a role in a shared name because it cannot open any session over any shared name. We also exclude processes like $\mathsf{for}\ x\ \mathsf{in}\ \ell\ \mathsf{do}\ Q$, which could potentially open several sessions of a global type (once in any iteration of the loop). The condition for $\mathsf{repeat}\ Q_1\ \mathsf{until}\ Q_2$ is analogous when requiring $u \notin \mathtt{fn}(Q_1)$.

To introduce the notion of implementations of a global type it is convenient to use *contexts*, that is terms derived from the following productions:

$$\mathtt{C}[\_] ::= \_ \ \mid\ \mathtt{C}[\_]|S \ \mid\ \ \mid\ S|\mathtt{C}[\_] \ \mid\ (\nu\vec{\mathsf{y}}@u)\mathtt{C}[\_]$$

Definition 6.2 (Implementation)**.** Let ι be a mapping assigning a process to each $\mathtt{p} \in \{\mathtt{p}_0, \ldots, \mathtt{p}_n\} \subseteq \mathbb{P}$, $P = \iota(\mathtt{p}_0) \mid \ldots \mid \iota(\mathtt{p}_n)$, $\vec{\mathsf{y}}$ a tuple of pairwise disjoint session channels in $\mathbb{Y}$, and $u \in \mathbb{U}$. A system $\iota_{\vec{\mathsf{y}}@u}$ is an *ι-implementation of $\vec{\mathsf{y}}$ at u for $\{\mathtt{p}_0, \ldots, \mathtt{p}_n\}$* if there is a context $\mathtt{C}[\_] = (\nu\vec{\mathsf{y}}_1@u_1)\cdots(\vec{\mathsf{y}}_h@u_h)(\_ \mid S)$ such that

(1) $u \notin \{u_1, \ldots u_h\}$, $(\{u\} \cup \vec{\mathsf{y}}) \cap \mathtt{fn}(S) = \emptyset$, and $\vec{\mathsf{y}} \cap \left(\bigcup_{i=1,\ldots,h} \vec{\mathsf{y}}_i\right) = \emptyset$
(2) if $\iota_{\vec{\mathsf{y}}@u} \equiv \mathtt{C}[P]$ then, for all $0 \le j \le n$, $\iota(\mathtt{p}_j)$ uniquely plays role $\mathtt{p}_j$ in u
(3) if $\iota_{\vec{\mathsf{y}}@u} \equiv \mathtt{C}[(\nu\vec{\mathsf{y}}@u)(P) \mid \vec{\mathsf{y}}[\vec{\mathsf{v}}_1, \ldots, \vec{\mathsf{v}}_k]]$ for some $\vec{\mathsf{v}}_1, \ldots, \vec{\mathsf{v}}_k$ then $u \notin \mathtt{fn}(P)$.

Given a global type $\mathcal{G}(\vec{\mathsf{y}})$, an *$\iota$-implementation of $\mathcal{G}(\vec{\mathsf{y}})$ at u* is an ι-implementation at u for $\mathcal{P}(\mathcal{G})$.

Intuitively, a system $\iota_{\vec{\mathsf{y}}@u}$ is an implementation of $\mathcal{G}(\vec{\mathsf{y}})$ if $\iota_{\vec{\mathsf{y}}@u}$ is built-up from processes that implement all the roles in $\mathcal{G}(\vec{\mathsf{y}})$; the association between roles and processes is given by the function ι. In addition, $\iota_{\vec{\mathsf{y}}@u}$ may contain other processes, possibly running different sessions. Technically, we require $\iota_{\vec{\mathsf{y}}@u}$ to be written in terms of a context $\mathtt{C}[\_] = (\nu\vec{\mathsf{y}}_1@u_1)\cdots(\vec{\mathsf{y}}_h@u_h)(\_ \mid S)$, which describes the part of the system that does not directly implement $\mathcal{G}(\vec{\mathsf{y}})$. The conditions $u \notin \{u_1, \ldots u_h\}$, $\vec{\mathsf{y}} \cap \left(\bigcup_{i=1,\ldots,h} \vec{\mathsf{y}}_i\right) = \emptyset$, and $(\{u\} \cup \vec{\mathsf{y}}) \cap \mathtt{fn}(S) = \emptyset$ ensure that the context does not interfere with the names used for implementing $\mathcal{G}(\vec{\mathsf{y}})$. Then, an implementation has two different shapes depending on whether the session for $\mathcal{G}(\vec{\mathsf{y}})$ has been initiated or not. Condition (2) stipulates that, before starting the session, each process $\iota(\mathtt{p})$ uniquely plays the role $\mathtt{p}$ in u (i.e., $\iota(\mathtt{p})$ is able to open a session on u for the role $\mathtt{p}$). Condition (3) characterises the case in which the session has been initiated, and therefore the system contains the message queues for the initiated session.

Example 6.3. Consider the global type $\mathsf{G}_{\mathsf{POP}}$ in Section 3 and take $\iota_{\vec{\mathsf{y}}@u} = \iota(\mathsf{s}) \mid \iota(\mathsf{c})$ where ι is such that $\iota(\mathsf{s}) = \mathit{Init}$ and $\iota(\mathsf{c}) = C$ with Init the process in (5.1) (*cf.* page 16) and $C \triangleq \overline{u}^1(\vec{\mathsf{y}}).\overline{\mathsf{quit}}\,;\mathsf{bye}$. It is easy to check that Init uniquely plays s in u while C uniquely plays c (after assuming that c is 0). Hence, it is straightforward that $\iota_{\vec{\mathsf{y}}@u}$ is a *ι-implementation* of $\mathcal{G}(\vec{\mathsf{y}}) = \mathsf{G}_{\mathsf{POP}}$ at u (it is enough to consider the identity context $\mathtt{C}[\_] = \_$).

Consider now a more involved situation in which a process C' that implements c also interacts with another process D over a different session, e.g.,

$$C' \triangleq u_0{}^{\mathsf{a}}(\mathsf{z}).\overline{u}^1(\vec{\mathsf{y}}).\overline{\mathsf{quit}};\mathsf{bye}.C'' \qquad D \triangleq \overline{u_0}^1(\mathsf{z}).D'$$

In this case, ι' is such that $\iota'(\mathsf{s}) = \mathit{Init}$ and $\iota'(\mathsf{c}) = C'$. Then, $\iota'_{\vec{\mathsf{y}}@u} = \mathit{Init} \mid C \mid D$ is an ι'-*implementation* of $\mathcal{G}(\vec{\mathsf{y}})$ at u (it suffices to consider the context $\mathtt{C}[\_] = \_ \mid D$). Note that

$$\iota'_{\vec{\mathsf{y}}@u} \xrightarrow{\tau} (\nu \mathsf{z}@u_0)(\mathit{Init} \mid C''' \mid D' \mid \vec{\mathsf{z}}[]) = \iota''_{\vec{\mathsf{y}}@u}$$

with $C''' \triangleq \overline{u}^1(\vec{\mathsf{y}}).\overline{\mathsf{quit}};\mathsf{bye}.C''$ and $\iota'' = \iota'[\mathsf{c} \mapsto C''']$. Then, we can conclude that $\iota''_{\vec{\mathsf{y}}@u}$ is an ι''-implementation of $\mathcal{G}(\vec{\mathsf{y}})$ at u by considering the context $\mathtt{C}[\_] = (\nu \mathsf{z}@u_0)(\_ \mid D' \mid \vec{\mathsf{z}}[])$.

Consider now the transition

$$\iota''_{\vec{\mathsf{y}}@u} \xrightarrow{\tau} (\nu \mathsf{z}@u_0)((\nu \vec{\mathsf{y}}@u)(\mathit{Srv} \mid C'''' \mid \vec{\mathsf{y}}[[] \dots []]) \mid D' \mid \mathsf{z}[]) = \iota'''_{\vec{\mathsf{y}}@u}$$

where $\iota'''(\mathsf{s}) = \mathit{Srv}$ and $\iota'''(\mathsf{c}) = C''''$. In this case $\iota'''_{\vec{\mathsf{y}}@u}$ is an ι'''-implementation of $\mathcal{G}(\vec{\mathsf{y}})$ at u; the sub-term $(\nu \vec{\mathsf{y}}@u)(\mathit{Srv} \mid C'''' \mid \vec{\mathsf{y}}[[] \dots []])$ stands for the session corresponding to the global type $\mathcal{G}(\vec{\mathsf{y}})$, while the context represents the rest of the system. ◇

We characterise WSI as a relation between the execution traces of a global type $\mathcal{G}$ and its implementations $\iota_{\vec{\mathsf{y}}@u}$. An execution trace of a system $\iota_{\vec{\mathsf{y}}@u}$ is a sequence of events of the form $\langle \mathsf{p}, \overline{\mathsf{y}}\ \mathsf{d} \rangle$ and $\langle \mathsf{p}, \mathsf{y}\ \mathsf{d} \rangle$, which respectively represent an output and an input action performed by p over the channel y.

Definition 6.4 (Runs of implementations)**.** Let $\iota_{\vec{\mathsf{y}}@u}$ be an ι-implementation of $\mathcal{G}(\vec{\mathsf{y}})$. The set of *runs of* $\iota_{\vec{\mathsf{y}}@u}$ *initiated on* u *with store* σ, written $\mathcal{R}\langle \iota_{\vec{\mathsf{y}}@u}, \sigma \rangle$, is the set inductively defined by the rules in Figure 6. We write $\mathcal{R}\iota_{\vec{\mathsf{y}}@u}$ for $\mathcal{R}\langle \iota_{\vec{\mathsf{y}}@u}, \emptyset \rangle$ and extend the notion to sets of implementations $\mathbb{I}$ as $\mathcal{R}\mathbb{I} = \cup_{I \in \mathbb{I}} \mathcal{R}I$.

Rules in Figure 6 rely on the semantics of Figure 4 and Figure 5. Rule [REnd] establishes that a completed session, i.e., one in which all processes are terminated and the session queues are empty, contains the empty run ϵ. Non-empty runs of $\iota_{\vec{\mathsf{y}}@u}$ are defined in terms of the input and output actions that processes $\iota(\mathsf{p})$ perform over the session channels $\vec{\mathsf{y}}$, as described by the rules [RSnd] and [RRcv]. In rule [RSnd], $\iota(\mathsf{p})$ performs an output over a session channel associated with u; which is formally captured by the conditions $\langle \iota(\mathsf{p}), \sigma \rangle \xrightarrow{e \vdash \overline{\mathsf{y}}\mathsf{v}} \langle P, \sigma \rangle$ and $\mathsf{y} \in \vec{\mathsf{y}}$ in the premiss. When $\iota(\mathsf{p})$ evolves to P by performing $\overline{\mathsf{y}}\mathsf{v}$, $\iota_{\vec{\mathsf{y}}@u}$ evolves to $\iota'_{\vec{\mathsf{y}}@u}$ with $\iota' = \iota[\mathsf{p} \mapsto P]$, i.e., ι' coincides with ι in all roles but p. This is stated by the condition $\langle \iota_{\vec{\mathsf{y}}@u}, \sigma \rangle \xrightarrow{e' \vdash \tau} \langle \iota[\mathsf{p} \mapsto P]_{\vec{\mathsf{y}}@u}, \sigma \rangle$ in the premiss of the rule. Hence, $\iota_{\vec{\mathsf{y}}@u}$ contains a run $\langle \mathsf{p}, \overline{\mathsf{y}}\mathsf{d} \rangle r$ (see the conclusion of the rule) when r is a run of the state $\langle \iota[\mathsf{p} \mapsto P]_{\vec{\mathsf{y}}@u}, \sigma' \rangle$ reached after $\iota(\mathsf{p})$ performs $\overline{\mathsf{y}}\mathsf{v}$. We remark that runs abstract away from the particular values sent by the processes and keep instead the sorts of sent value (i.e., condition $\mathsf{v} : \mathsf{d}$). Input events are handled analogously in rule [RRcv]; in this case also σ evolves to σ' when $\iota(\mathsf{p})$ performs an input.

Rule [RExt_1] accounts for the computation steps of $\iota(\mathsf{p})$ that do not involve session channels in $\vec{\mathsf{y}}$ (condition $\mathtt{n}(\alpha) \cap \vec{\mathsf{y}} = \emptyset$), which can be an internal transition τ in a role, a communication over a channel not in $\vec{\mathsf{y}}$, or a session initiation. This rule allows each process to freely initiate a session that does not correspond to the global type G, i.e., over a shared name different from u (condition $u \notin \mathtt{fn}(\beta)$). Rule [$\mathsf{RExt}_2$] handles the cases in which the transition of $\iota_{\vec{\mathsf{y}}@u}$ does not involve any process $\iota(\mathsf{p})$. This is captured by the fact that the continuation $\iota'_{\vec{\mathsf{y}}@u}$ uses the same mapping ι. By the definition of $\iota_{\vec{\mathsf{y}}@u}$, the reduction does not interfere with the names of the session, i.e., $\mathtt{fn}(\beta) \cap (\vec{\mathsf{y}} \cup \{u\})$ holds.

$$\text{[REnd]}\quad \frac{\forall \mathsf{p} \in \mathcal{P}(\mathcal{G}) : \ \iota(\mathsf{p}) = 0 \qquad \forall \mathsf{y} \in \vec{\mathsf{y}} : \ \text{queue on } \mathsf{y} \text{ is empty in } \iota_{\vec{\mathsf{y}}@u}}{\epsilon \in \mathcal{R}\langle \iota_{\vec{\mathsf{y}}@u}, \sigma\rangle}$$

$$\text{[RSnd]}\quad \frac{\langle \iota(\mathsf{p}), \sigma\rangle \xrightarrow{e \vdash \overline{\mathsf{y}}\mathsf{v}} \langle P, \sigma\rangle \qquad \mathsf{y} \in \vec{\mathsf{y}} \qquad \vdash \mathsf{v} : \mathsf{d} \qquad \langle \iota_{\vec{\mathsf{y}}@u}, \sigma\rangle \xrightarrow{e \vdash \tau} \langle \iota[\mathsf{p} \mapsto P]_{\vec{\mathsf{y}}@u}, \sigma\rangle \qquad r \in \mathcal{R}\langle \iota[\mathsf{p} \mapsto P]_{\vec{\mathsf{y}}@u}, \sigma\rangle}{\langle \mathsf{p}, \overline{\mathsf{y}}\mathsf{d}\rangle r \in \mathcal{R}\langle \iota_{\vec{\mathsf{y}}@u}, \sigma\rangle}$$

$$\text{[RRcv]}\quad \frac{\langle \iota(\mathsf{p}), \sigma\rangle \xrightarrow{e \vdash \mathsf{y}\mathsf{v}} \langle P, \sigma'\rangle \qquad \mathsf{y} \in \vec{\mathsf{y}} \qquad \vdash \mathsf{v} : \mathsf{d} \qquad \langle \iota_{\vec{\mathsf{y}}@u}, \sigma\rangle \xrightarrow{e \vdash \tau} \langle \iota[\mathsf{p} \mapsto P]_{\vec{\mathsf{y}}@u}, \sigma'\rangle \qquad r \in \mathcal{R}\langle \iota[\mathsf{p} \mapsto P]_{\vec{\mathsf{y}}@u}, \sigma'\rangle}{\langle \mathsf{p}, \mathsf{y}\mathsf{d}\rangle r \in \mathcal{R}\langle \iota_{\vec{\mathsf{y}}@u}, \sigma\rangle}$$

$$\text{[REXT}_1\text{]}\quad \frac{\langle \iota(\mathsf{p}), \sigma\rangle \xrightarrow{e \vdash \alpha} \langle P, \sigma'\rangle \qquad \mathtt{n}(\alpha) \cap \vec{\mathsf{y}} = \emptyset \qquad \langle \iota_{\vec{\mathsf{y}}@u}, \sigma\rangle \xrightarrow{e' \vdash \beta} \langle \iota[\mathsf{p} \mapsto P]_{\vec{\mathsf{y}}@u}, \sigma'\rangle \qquad u \notin \mathtt{fn}(\beta) \qquad r \in \mathcal{R}\langle \iota[\mathsf{p} \mapsto P]_{\vec{\mathsf{y}}@u}, \sigma'\rangle}{r \in \mathcal{R}\langle \iota_{\vec{\mathsf{y}}@u}, \sigma\rangle}$$

$$\text{[REXT}_2\text{]}\quad \frac{\langle \iota_{\vec{\mathsf{y}}@u}, \sigma\rangle \xrightarrow{e' \vdash \beta} \langle (\iota_{\vec{\mathsf{y}}@u})', \sigma'\rangle \qquad r \in \mathcal{R}\langle (\iota_{\vec{\mathsf{y}}@u})', \sigma'\rangle}{r \in \mathcal{R}\langle \iota_{\vec{\mathsf{y}}@u}, \sigma\rangle}$$

$$\text{[ROpen]}\quad \frac{\langle \iota(\mathsf{p}_0), \sigma\rangle \xrightarrow{e_0 \vdash \overline{u}^{\mathsf{n}}(\vec{\mathsf{y}})} \langle P_0, \sigma_0\rangle \qquad \forall 1 \le i \le n : \ \langle \iota(\mathsf{p}_i), \sigma\rangle \xrightarrow{e_i \vdash u^{\mathsf{i}}(\vec{\mathsf{y}})} \langle P_i, \sigma_i\rangle \qquad \mathcal{P}(\mathcal{G}(\vec{\mathsf{y}})) = \{\mathsf{p}_0, \mathsf{p}_1, \ldots, \mathsf{p}_n\} \qquad \iota' = \iota[\mathsf{p}_0 \mapsto P_0] \ldots [\mathsf{p}_n \mapsto P_n] \qquad \langle \iota_{\vec{\mathsf{y}}@u}, \sigma\rangle \xrightarrow{e \vdash \tau} \langle \iota'_{\vec{\mathsf{y}}@u}, \sigma[\vec{\mathsf{y}} \mapsto u]\rangle \qquad r \in \mathcal{R}\langle \iota'_{\vec{\mathsf{y}}@u}, \sigma[\vec{\mathsf{y}} \mapsto u]\rangle}{r \in \mathcal{R}\langle \iota_{\vec{\mathsf{y}}@u}, \sigma\rangle}$$

Figure 6: Runs of implementations

Rule [ROpen] allows for the initiation of a new session on u and requires all roles to participate in the synchronisation (as stated by the three first premisses). We assume that any role in the implementation will execute exactly one action over the channel u which also matches the role assigned by ι. Nested sessions are handled by assuming that all sessions are created over different channels that have the same type. This is just a technical simplification analogous to the possibility of having annotations to indicate the particular instance of the session under analysis.

The runs of an implementation abstractly capture the traces of communications of the processes in the system. This can be easily formalised by using a more concrete relation on systems. More precisely, we define $\hookrightarrow$ as the relation induced by the rules such as those in

$$[\mathsf{RGEnd}]\quad \epsilon \in \mathcal{R}(\mathsf{end}) \qquad [\mathsf{RGCom}]\quad \frac{j \in I \qquad r \in \mathcal{R}(\mathsf{G}_j)}{\langle \mathsf{p}, \overline{\mathsf{y}}_j \mathsf{d}_j \rangle \langle \mathsf{q}_j, \mathsf{y}_j \mathsf{d}_j \rangle r \in \mathcal{R}(\sum_{i \in I} \mathsf{p} \twoheadrightarrow \mathsf{q}_i : \mathsf{y}_i\ \mathsf{d}_i; \mathsf{G}_i)}$$

$$[\mathsf{RGSeq}]\quad \frac{r_1 \in \mathcal{R}(\mathsf{G}_1) \quad r_2 \in \mathcal{R}(\mathsf{G}_2)}{r_1 r_2 \in \mathcal{R}(\mathsf{G}_1; \mathsf{G}_2)} \qquad [\mathsf{RG}_1^*]\quad \frac{r \in \mathcal{R}(\mathsf{G})}{r \in \widetilde{\mathcal{R}}(\mathsf{G}^{*^f})} \qquad [\mathsf{RG}_2^*]\quad \frac{r_1 \in \mathcal{R}(\mathsf{G}) \quad r_2 \in \widetilde{\mathcal{R}}(\mathsf{G}^{*^f})}{r_1[r_2] \in \widetilde{\mathcal{R}}(\mathsf{G}^{*^f})}$$

$$[\mathsf{RGIter}]\quad \frac{r \in \widetilde{\mathcal{R}}(\mathsf{G}^{*^f}) \quad \mathtt{rdy}(\mathsf{G}) = \{\mathsf{p}\} \quad \mathcal{P}(\mathsf{G}) = \{\mathsf{p}, \mathsf{p}_1, \ldots, \mathsf{p}_n\} \quad \forall 1 \leq i \leq n : \ f(\mathsf{p}_i) = \mathsf{y}_i \mathsf{d}_i}{r \langle \mathsf{p}, \overline{\mathsf{y}_1} \mathsf{d}_1 \rangle \langle \mathsf{p}_1, \mathsf{y}_1 \mathsf{d}_1 \rangle \ldots \langle \mathsf{p}, \overline{\mathsf{y}_n} \mathsf{d}_n \rangle \langle \mathsf{p}_n, \mathsf{y}_n \mathsf{d}_n \rangle \quad \in \quad \mathcal{R}(\mathsf{G}^{*^f})}$$

Figure 7: Runs of a global type

Figure 4 once the τ in the conclusion of rules [SCom_1] and [SCom_2] is replaced with the output and input action respectively. Then we can state the following proposition:

Theorem 6.5. *Given an implementation $\iota_{\vec{\mathsf{y}}@u}$ and a store σ, if $r \in \mathcal{R}\langle \iota_{\vec{\mathsf{y}}@u}, \sigma \rangle$ is a run of length m then there is a sequence $\iota_{\vec{\mathsf{y}}@u} \xhookrightarrow{\alpha_1} \langle S_1, \sigma_1 \rangle \cdots \xhookrightarrow{\alpha_n} \langle S_n, \sigma_n \rangle$ such that we can find an order preserving bijection χ between the sets $\{1, \ldots, m\}$ and $\{1 \leq j \leq n \mid \alpha_i \text{ not a } \tau\}$ such that the i-th element in r is an input of sort d iff so is the $\alpha_{\chi(i)}$ and the value in $\alpha_{\chi(i)}$ has the sort d.*

Proof. Straightforward induction on the derivation of $r \in \mathcal{R}\langle \iota_{\vec{\mathsf{y}}@u}, \sigma \rangle$. □

We now introduce the notion of runs associated to a global type. Our notion of WSI will allow us to implement an iterative type, which accounts for an unbounded number of repetitions, with a process exhibiting a bounded number of iterations. For this reason, we deviate from the previous definition of traces of global types [14, 10, 25, 24] and use *annotated traces* to distinguish mandatory from optional events. Annotating optional events is instrumental to the comparison of traces of iterative types (which is defined below). Syntactically, an optional sequence r of events is written $[r]$. As usual, we consider an asynchronous communication model and a trace implicitly denotes the equivalence class of all traces obtained by permuting causally independent events, that is events executed by different participants on different channels.

Definition 6.6 (Runs of a global type)**.** Given a global type term G, the set $\mathcal{R}(\mathsf{G})$ denotes the runs allowed by G and is defined as the least set closed under the rules in Figure 7.

The first three rules of Figure 7 are straightforward. The runs of an iterative type G^{*^f} are given by the rule [RGIter], whose premiss uses the set $\widetilde{\mathcal{R}}(\mathsf{G}^{*^f})$ to unfold G^{*^f} (as defined by the rules [RG_1^*] and [RG_2^*]). Optional events are introduced when unfolding an iterative type (rule [RG_2^*]). The main motivation is that an iterative type G^{*^f} denotes an unbounded number of repetitions of (traces of) G (i.e., an infinite number of traces). Note that $\widetilde{\mathcal{R}}(\mathsf{G}^{*^f}) = \{r_1, r_1[r_2], r_1[r_2[r_3]], \ldots\}$ with $r_i \in \mathcal{R}(\mathsf{G})$. Rule [$\mathsf{RGIter}$] adds the events associated to the termination of an iteration: (i) the ready role p sends the termination signal to any other role by using the dedicated channels specified by f (i.e., $\langle \mathsf{p}, \overline{\mathsf{y}_1} \mathsf{d}_1 \rangle, \ldots, \langle \mathsf{p}, \overline{\mathsf{y}_n} \mathsf{d}_n \rangle$), and (ii) all roles but the ready one receive the corresponding termination message (i.e.,

$\langle \mathsf{p}_1, \mathsf{y}_1\mathsf{d}_1\rangle, \ldots, \langle \mathsf{p}_n, \mathsf{y}_n\mathsf{d}_n\rangle$). We just consider one of the possible interleavings of termination events because we consider traces up-to the permutation of causally independent events.

Definition 6.7 (Trace preorder)**.** The *trace preorder* $\preccurlyeq$ is the least preorder on annotated traces satisfying the following axioms and rules.

$$\frac{}{[r] \preccurlyeq \epsilon}\ [\preccurlyeq\mathsf{drop}] \qquad \frac{}{\epsilon \preccurlyeq r}\ [\preccurlyeq\mathsf{emp}] \qquad \frac{r_1 \preccurlyeq r_1' \quad r_2 \preccurlyeq r_2'}{r_1 r_2 \preccurlyeq r_1' r_2'}\ [\preccurlyeq\mathsf{cmp}]$$

We say r' *covers* r when $r \preccurlyeq r'$, i.e., when r' matches all mandatory actions of r. Analogously, we say a set R_2 of annotated traces *covers* another set R_1, written $R_1 \Subset R_2$, if for all $r \in R_1$ there is $r' \in R_2$ such that $r \preccurlyeq r'$.

Definition 6.8 (Whole-spectrum implementation)**.** A set $\mathbb{I}$ of implementations *covers* a global type $\mathcal{G}(\vec{\mathsf{y}}) \triangleq \mathsf{G}$ if $\mathcal{R}(\mathsf{G}) \Subset \mathcal{R}\mathbb{I}$. A process P is a *whole-spectrum implementation of* $\mathsf{p} \in \mathcal{P}(\mathsf{G})$ when there exists a set $\mathbb{I}$ of ι-implementations of $\mathcal{G}(\vec{\mathsf{y}})$ at u that covers G such that $\iota_{\vec{\mathsf{y}}@u} \in \mathbb{I}$ implies $\iota(\mathsf{p}) = P$.

A whole-spectrum implementation (WSI) of a role p is a process P such that any expected behaviour of the global type can be obtained by putting P into a proper context. For iteration types, the comparison of annotated traces implies that the implementation has to be able to perform the iteration body at least once, but it can arbitrarily choose the number of iterations.

Remark 6.9 (Trace semantics for WSI)**.** WSI is based on a trace semantics. WSI implementations are deterministic programs, hence their behaviour if faithfully captured by the set of their execution traces. A stronger notion like bisimulation is not necessary. In fact, the branching specified in a global type becomes deterministic in a process, therefore the set of all possible executions of a process (in a given implementation) results in a single trace.

7. Typing

In this section we introduce a typing discipline to guarantee that a well-typed process is a WSI of the role it plays in a global type. Technically we rely on an enriched version of local types, dubbed *pseudo-types*, that takes into account branching enabling conditions.

7.1. Pseudo-types & Typing judgements.

The scaffolding of our typing discipline is standard but for the need of making the typing depending on the expressions the processes use to render choices. This requires to revisit the usual definition of mergeability (cf. Definition 7.1 below) that now relies on a notion of *normalisation* of local types.

The syntax of pseudo-types is given by the following grammar:

$$\mathbb{T} ::= \bigoplus_{i\in I} e_i \prec \overline{\mathsf{y}_i}\ \mathsf{d}_i.\mathbb{T}_i \quad \mid \quad \sum_{i\in I} e_i \prec \mathsf{y}_i\ \mathsf{d}_i.\mathbb{T}_i \quad \mid \quad \mathbb{T}_1;\mathbb{T}_2 \quad \mid \quad \mathbb{T}^\star \quad \mid \quad e \prec \mathsf{end}$$

We call *guards* the expressions e_i occurring in a pseudo-type. Guards keep track of the conditions that have to be satisfied in order to enable a certain behaviour. For instance, the pseudo-type

$$\mathbb{T}_\oplus = \bigoplus_{i\in\{1,2\}} e_i \prec \overline{\mathsf{y}_i}\ \mathsf{int}.\mathsf{end}$$

$$
\begin{array}{lrcl}
(1) & \mathsf{nf}(e, e' \prec \mathsf{end}) & = & e \wedge e' \prec \mathsf{end} \\
\multicolumn{4}{l}{\text{let } J = \{i \in I \mid (e \wedge e_i) \iff \mathtt{false}\} \text{ in the clauses (2) and (3) below} } \\
(2) & \mathsf{nf}(e, \bigoplus_{i\in I} e_i \prec \overline{\mathsf{y}_i}\ \mathsf{d}_i.\mathbb{T}_i) & = & \begin{cases} \mathtt{false} \prec \mathsf{end} & \text{if } I = J \\ \bigoplus_{i\in I\setminus J} e_i \wedge e \prec \overline{\mathsf{y}_i}\ \mathsf{d}_i.\mathsf{nf}(e_i \wedge e, \mathbb{T}_i) & \text{if } I \neq J \end{cases} \\
(3) & \mathsf{nf}(e, \sum_{i\in I} e_i \prec \mathsf{y}_i\ \mathsf{d}_i.\mathbb{T}_i) & = & \begin{cases} \mathtt{false} \prec \mathsf{end} & \text{if } I = J \\ \sum_{i\in I\setminus J} e_i \wedge e \prec \mathsf{y}_i\ \mathsf{d}_i.\mathsf{nf}(e_i \wedge e, \mathbb{T}_i) & \text{if } I \neq J \end{cases} \\
(4) & \mathsf{nf}(e, e' \prec \mathsf{end}; \mathbb{T}) & = & \mathsf{nf}(e \wedge e', \mathbb{T}) \\
(5) & \mathsf{nf}(e, (\bigoplus_{i\in I} e_i \prec \overline{\mathsf{y}_i}\ \mathsf{d}_i.\mathbb{T}_i); \mathbb{T}) & = & \mathsf{nf}(e, \bigoplus_{i\in I} e_i \prec \overline{\mathsf{y}_i}\ \mathsf{d}_i.(\mathbb{T}_i; \mathbb{T})) \\
(6) & \mathsf{nf}(e, (\sum_{i\in I} e_i \prec \mathsf{y}_i\ \mathsf{d}_i.\mathbb{T}_i); \mathbb{T}) & = & \mathsf{nf}(e, \sum_{i\in I} e_i \prec \mathsf{y}_i\ \mathsf{d}_i.(\mathbb{T}_i; \mathbb{T})) \\
(7) & \mathsf{nf}(e, (\mathbb{T}; \mathbb{T}'); \mathbb{T}'') & = & \mathsf{nf}(e, \mathbb{T}; (\mathbb{T}'; \mathbb{T}'')) \\
(8) & \mathsf{nf}(e, \mathbb{T}^\star; \mathbb{T}') & = & \begin{cases} e' \prec \mathsf{end} & \text{if } \mathsf{nf}(e, \mathbb{T}) = e' \prec \mathsf{end} \\ \mathsf{nf}(e, \mathbb{T}^\star); \mathsf{nf}(e, \mathbb{T}') & \text{if } \mathsf{nf}(e, \mathbb{T}) \neq e' \prec \mathsf{end} \end{cases} \\
(9) & \mathsf{nf}(e, \mathbb{T}^\star) & = & \begin{cases} e' \prec \mathsf{end} & \text{if } \mathsf{nf}(e, \mathbb{T}) = e' \prec \mathsf{end} \\ \mathsf{nf}(e, \mathbb{T})^\star & \text{if } \mathsf{nf}(e, \mathbb{T}) \neq e' \prec \mathsf{end} \end{cases}
\end{array}
$$

Figure 8: Normalisation procedure for pseudo-types

where the guards are $e_1 = x > 0$ and $e_2 = x \leq 0$. By e_1, x needs to be strictly positive in order to choose the first branch. Local types from Section 2 can be thought of as pseudo-types where all guards are `true`. Hereafter we may omit guards `true` and have e.g.,

$\mathtt{true} \prec \mathsf{end}$ abbreviated as end and $\bigoplus_{i\in I} \mathtt{true} \prec \overline{\mathsf{y}_i}\ \mathsf{d}_i.\mathbb{T}_i$ abbreviated as $\bigoplus_{i\in I} \overline{\mathsf{y}_i}\ \mathsf{d}_i.\mathbb{T}_i$

The notions of free and bound names straightforwardly extend to pseudo-types. We will write $\mathrm{var}(\mathbb{T})$ for the set of variables occurring in the expressions of $\mathbb{T}$ and $\mathtt{fy}(\mathbb{T})$ for the set of session channels in $\mathbb{T}$; for instance, $\mathrm{var}(\mathbb{T}_\oplus) = \{x\}$.

Given a pseudo-type $\mathbb{T}$, the *normal form of* $\mathbb{T}$, written $\mathsf{nf}(\mathbb{T})$, is defined as $\mathsf{nf}(\mathbb{T}) = \mathsf{nf}(\mathtt{true}, \mathbb{T})$ where $\mathsf{nf}(\_,\_)$ given by the equations in Figure 8. Intuitively, the normalisation of a pseudo-type propagates the guards of branches to their continuations while removing those alternatives with inconsistent guards. We just remark that $\mathsf{nf}(\mathbb{T})$ is defined for any $\mathbb{T}$ (details are in Appendix A.1).

The notion of normalisation is instrumental to adapt the standard merge operation $\_\bowtie\_$ of session types [10] to pseudo-types. Our definition of $\_\bowtie\_$ requires the *mergeability* of pseudo-types, which amounts to have branches with the same communication prefix guarded by mutually exclusive conditions.

Definition 7.1 (Mergeable pseudo-types)**.** Two pseudo-types $\mathbb{T}_1$ and $\mathbb{T}_2$ in normal form are *mergeable*, if

- $\mathbb{T}_1 = e \prec \mathsf{end}$ and $\mathbb{T}_2 = e' \prec \mathsf{end}$
- $\mathbb{T}_1 = \sum_{i\in I} e_i \prec \mathsf{y}_i\ \mathsf{d}_i.\mathbb{T}_i$, $\mathbb{T}_2 = \sum_{i\in I} e'_i \prec \mathsf{y}_i\ \mathsf{d}_i.\mathbb{T}'_i$ and for all $i \in I$, $\mathbb{T}_i$ and $\mathbb{T}'_i$ are mergeable, and $e_i \wedge e'_i \iff \mathtt{false}$

- $\mathbb{T}_1 = \bigoplus_{i\in I\cup J} e_i \prec \overline{\mathsf{y}_i}\ \mathsf{d}_i.\mathbb{T}_i$ and $\mathbb{T}_2 = \bigoplus_{i\in I\cup K} e'_i \prec \overline{\mathsf{y}_i}\ \mathsf{d}_i.\mathbb{T}'_i$ with $I\cap J = I\cap K = \emptyset$ and sets $\{\mathsf{y}_i\}_{i\in I}$, $\{\mathsf{y}_j\}_{j\in J}$, and $\{\mathsf{y}_k\}_{k\in K}$ pairwise disjoint, and for all $i\in I$, $\mathbb{T}_i$ and $\mathbb{T}'_i$ are mergeable, and $e_i \wedge e'_i \iff \texttt{false}$
- $\mathbb{T}_1 = \mathbb{T}'_1;\mathbb{T}''_1$, $\mathbb{T}_2 = \mathbb{T}'_2;\mathbb{T}''_2$, and $\mathbb{T}'_1$ and $\mathbb{T}'_2$ as well as $\mathbb{T}''_1$ and $\mathbb{T}''_2$ mergeable
- $\mathbb{T}_1 = (\mathbb{T}'_1)^\star$, $\mathbb{T}_2 = (\mathbb{T}'_2)^\star$ with $\mathbb{T}'_1$ and $\mathbb{T}'_2$ mergeable.

Basically, $\mathbb{T}_1$ and $\mathbb{T}_2$ are mergeable when they have the same structure and at choice points branches either use different channels or they use mutually exclusive guards. When types are mergeable, operation $\_\bowtie\_$ "glues" branches that use the same channel.

Definition 7.2 (Merge)**.** The *merge* $\mathbb{T}_1\bowtie\mathbb{T}_2$ of two mergeable pseudo-types $\mathbb{T}_1$ and $\mathbb{T}_2$ is defined as:

$$\mathbb{T}_1\bowtie\mathbb{T}_2 = \begin{cases} e\vee e' \prec \mathsf{end} & \text{if } \mathbb{T}_1 = e\prec\mathsf{end} \text{ and } \mathbb{T}_2 = e'\prec\mathsf{end} \\ \sum_{i\in I} e_i\vee e'_i \prec \mathsf{y}_i\ \mathsf{d}_i.\mathbb{T}'_i\bowtie\mathbb{T}'_i & \text{if } \mathbb{T}_1 = \sum_{i\in I} e_i\prec \mathsf{y}_i\ \mathsf{d}_i.\mathbb{T}_i \text{ and } \mathbb{T}_2 = \sum_{i\in I} e'_i\prec \mathsf{y}_i\ \mathsf{d}_i.\mathbb{T}'_i \\ \bigoplus_{i\in I\cup J} e_i \prec \overline{\mathsf{y}_i}\ \mathsf{d}_i.\mathbb{T}_i & \text{if } \mathbb{T}_1 = \bigoplus_{i\in I} e_i \prec \overline{\mathsf{y}_i}\ \mathsf{d}_i.\mathbb{T}_i \text{ and } \mathbb{T}_2 = \bigoplus_{j\in J} e_j \prec \overline{\mathsf{y}_j}\ \mathsf{d}_j.\mathbb{T}_j \\ (\mathbb{T}'_1\bowtie\mathbb{T}'_2);(\mathbb{T}''_1\bowtie\mathbb{T}''_2) & \text{if } \mathbb{T}_1 = \mathbb{T}'_1;\mathbb{T}''_1 \text{ and } \mathbb{T}_2 = \mathbb{T}'_2;\mathbb{T}''_2 \\ (\mathbb{T}'_1\bowtie\mathbb{T}'_2)^\star & \text{if } \mathbb{T}_1 = (\mathbb{T}'_1)^\star \text{ and } \mathbb{T}_2 = (\mathbb{T}'_2)^\star \end{cases}$$

As we will see, our typing discipline keeps track of the assumptions (i.e., the guards) necessary to reach a particular point in the processes. In fact, systems are typed by judgements of the form

$$e \llcorner \Gamma \vdash S \triangleright \Delta \tag{7.1}$$

stipulating that, under the assumption e and the type assignment of variables Γ, the system S is typed as Δ. In (7.1), Γ and Δ are (possibly empty) partial functions. We adopt the usual syntactic notation for environments:

$$\Gamma ::= \emptyset \mid \Gamma, x : \mathsf{d} \qquad \Delta ::= \emptyset \mid \Delta, u : \mathcal{G} \mid \Delta, (\vec{\mathsf{y}},\mathsf{p}) : \mathbb{T} \mid \Delta, \mathsf{y} : [\mathsf{d}]$$

Environments Γ assign sorts d to variables x. Environments Δ, called *specifications*, map (i) shared names u to global types $\mathcal{G}$, (ii) participants' sessions $(\vec{\mathsf{y}},\mathsf{p})$ to pseudo-types $\mathbb{T}$, and (iii) session names y to queues of sorts d. As usual, we implicitly assume that in a judgement of the form (7.1) the following holds:

- $x\notin \mathrm{dom}(\Gamma)$ when writing $\Gamma, x : \mathsf{d}$, and
- $u\notin \mathrm{dom}(\Delta)$ when writing $\Delta, u : \mathcal{G}$ (likewise for participants' sessions and for sessions' queues).

For judgements of the form (7.1) we also assume that

- $\mathtt{fx}(e)\subseteq \mathrm{dom}(\Gamma)$ and $\mathtt{fn}(S)\subseteq \mathrm{dom}(\Delta)$, and
- in $\Delta, (\vec{\mathsf{y}},\mathsf{p}) : \mathbb{T}$ it is $\mathtt{fy}(\mathbb{T})\subseteq \vec{\mathsf{y}}$ and

$$\forall (\vec{\mathsf{y}_1},\mathsf{p}_1), (\vec{\mathsf{y}_2},\mathsf{p}_2)\in \mathrm{dom}(\Delta), \vec{\mathsf{y}}_1\cap\vec{\mathsf{y}}_2\neq\emptyset \implies \vec{\mathsf{y}}_1 = \vec{\mathsf{y}}_2 \tag{7.2}$$

Predicates

Notation	Arity	Description
Δ end-*only*	1	every session in Δ is terminated
Δ *active*	1	every session in Δ is an internal guarded-choice
Δ_1 and Δ_2 *passively compatible*	2	Δ_1 and Δ_2 agree on the termination of an iteration
Δ_1 and Δ_2 *independent*	2	disjoint sessions and agreement on shared names

Operations

Notation	Arity	Description
$\Delta_1;\Delta_2$	2	sequential composition
$\Delta_1 \cup \Delta_2$	2	union
$\Delta^\star$	1	closure
$\Delta\vert_{\_\vec{\mathsf{y}}}$	2	restriction of Δ to names not in $\vec{\mathsf{y}}$

Figure 9: Summary of operations and predicates for environments

Condition (7.2) states that a session channel can be used only in one session. We sometimes write $\vec{\mathsf{y}} \in \mathrm{dom}(\Delta)$ when there exists p such that $(\vec{\mathsf{y}},\mathsf{p}) \in \mathrm{dom}(\Delta)$. Similarly, $\mathsf{y} \in \mathrm{dom}(\Delta)$ stands for $\exists \vec{\mathsf{y}} \in \mathrm{dom}(\Delta) \,:\, \mathsf{y} \in \vec{\mathsf{y}}$. The extension of $\mathrm{var}(\_)$ and $\mathtt{fy}(\_)$ to environments is straightforward.

Our typing relies on an operation that recovers types from pseudo-types by removing guards. Formally, guard removal $\eth(\cdot)$ is defined as

$$\eth(e \prec \mathsf{end}) = \mathsf{end} \qquad \eth(\mathbb{T};\mathbb{T}') = \eth(\mathbb{T});\eth(\mathbb{T}') \qquad \eth(\mathbb{T}^\star) = \eth(\mathbb{T})^\star$$

$$\eth\Big(\sum_{i\in I} e_i \prec \mathsf{y}_i\ \mathsf{d}_i.\mathbb{T}_i\Big) = \sum_{\alpha\in I/_\sim} \mathsf{y}_\alpha\ \mathsf{d}_\alpha.\eth\big(\bowtie_{i\in\alpha}\mathbb{T}_i\big)$$
$$\eth\Big(\bigoplus_{i\in I} e_i \prec \overline{\mathsf{y}_i}\ \mathsf{d}_i.\mathbb{T}_i\Big) = \bigoplus_{\alpha\in I/_\sim} \overline{\mathsf{y}_\alpha}\ \mathsf{d}_\alpha.\eth\big(\bowtie_{i\in\alpha}\mathbb{T}_i\big)$$

where in the last two equations $\sim$ is the equivalence relation on I defined as $i \sim j \iff \mathsf{y}_i = \mathsf{y}_j$ and $\mathsf{y}_\alpha\mathsf{d}_\alpha = \mathsf{y}_i\mathsf{d}_i$ for $i \in \alpha$. Other auxiliary operations and predicates on environments are listed in Figure 9 and formally defined in the following section.

7.2. **Typing rules.** The typing rules for processes and systems are grouped in Figure 10 and Figure 11. For the sake of readability, we restate the typing rules as we comment them so to introduce notation and concepts appearing in the rules as we present them.

To type a request for a new session $\overline{u}^{\mathsf{n}}(\vec{\mathsf{y}}).P$ we use the following rule

$$[\mathsf{VReq}]\quad \frac{\Delta(u) \equiv \mathcal{G}(\vec{\mathsf{y}}) \qquad e \lrcorner \Gamma \vdash P \triangleright \Delta, (\vec{\mathsf{y}},0) : \mathbb{T} \qquad \eth\big(\mathsf{nf}(\mathbb{T})\big) = \mathsf{nf}(\mathcal{G}(\vec{\mathsf{y}})\restriction \mathsf{p})}{e \lrcorner \Gamma \vdash \overline{u}^{\mathsf{n}}(\vec{\mathsf{y}}).P \triangleright \Delta}$$

The premiss checks that the continuation P can be typed with Δ extended with an assignment of the pseudo-type $\mathbb{T}$ to the participant's session $(\vec{\mathsf{y}},\mathsf{p})$, for some $\mathbb{T}$ *matching* the projection of the global type $\Delta(u)$ on the corresponding role. Intuitively, the type obtained by removing guards from $\mathbb{T}$ coincides with the projection of the global type.

The rule [VAcc] for typing the acceptance for the p-th role $u^{\mathsf{p}}(\vec{\mathsf{y}}).P$ is defined analogously. An external choice is checked by

$$\text{[VRcv]}\quad \frac{\forall i \in I \,:\, \mathsf{y}_i \in \vec{\mathsf{y}} \qquad e \llcorner \Gamma, x_i : \mathsf{d}_i \;\vdash\; P_i \;\triangleright\; \Delta, (\vec{\mathsf{y}},\mathsf{p}) : \mathbb{T}_i}{e \llcorner \Gamma \;\vdash\; \sum_{i\in I} \mathsf{y}_i(x_i).P_i \;\triangleright\; \Delta, (\vec{\mathsf{y}},\mathsf{p}) : \sum_{i\in I} e \prec \mathsf{y}_i\ \mathsf{d}_i.\mathbb{T}_i}$$

that types each branch P_i against the respective continuation of the type $(\vec{\mathsf{y}},\mathsf{p}) : \mathbb{T}_i$ (once Γ is extended with the type assignment on the bound name x_i); the first condition in the premiss permits to branch only over a subset of the session channels.

An output $\overline{\mathsf{y}}\, e'$ and the idle process are respectively typed as follows

$$\text{[VSend]}\quad \frac{\Gamma \vdash e' : \mathsf{d} \qquad \mathsf{y} \in \vec{\mathsf{y}} \qquad \Delta \text{ end-only}}{e \llcorner \Gamma \;\vdash\; \overline{\mathsf{y}}\, e' \;\triangleright\; \Delta, (\vec{\mathsf{y}},\mathsf{p}) : e \prec \overline{\mathsf{y}}\ \mathsf{d}; e \prec \mathsf{end}} \qquad \text{[VEnd]}\quad \frac{\Delta \text{ end-only}}{e \llcorner \Gamma \;\vdash\; 0 \;\triangleright\; \Delta}$$

The expression e' in [VSend] has to be of the sort expected on channel y; moreover, no further actions should occur on session channels (rendered with the condition Δ end-*only* abbreviating $\forall(\vec{\mathsf{y}'},\mathsf{q}) \in \mathrm{dom}(\Delta) \,:\, \mathsf{nf}(\Delta(\vec{\mathsf{y}'},\mathsf{q})) = e \prec \mathsf{end}$); for the idle process we simply require Δ to map each session channel to the end type.

The typing of sequential compositions is handled by the rule

$$\text{[VSeq]}\quad \frac{e \llcorner \Gamma \;\vdash\; P_1 \;\triangleright\; \Delta_1 \qquad e \llcorner \Gamma \;\vdash\; P_2 \;\triangleright\; \Delta_2}{e \llcorner \Gamma \;\vdash\; P_1; P_2 \;\triangleright\; \Delta_1; \Delta_2}$$

that requires to decompose the specification into Δ_1 and Δ_2 to respectively type each part of the sequential composition. In the conclusion of the rule, the partial operation $\_;\_$ on specifications requires that $\mathrm{dom}(\Delta_2) \subseteq \mathrm{dom}(\Delta_1)$ and $\Delta_1|_{\mathbb{U}\cup\mathbb{Y}} = \Delta_2|_{\mathbb{U}\cup\mathbb{Y}}$ and it is defined as follows

$$\begin{aligned} (\Delta_1;\Delta_2)|_{\mathbb{U}\cup\mathbb{Y}} &= \Delta_1|_{\mathbb{U}\cup\mathbb{Y}} \\ (\Delta_1;\Delta_2)(\vec{\mathsf{y}},\mathsf{p}) &= \begin{cases} \Delta_1(\vec{\mathsf{y}},\mathsf{p}); \Delta_2(\vec{\mathsf{y}},\mathsf{p}) & (\vec{\mathsf{y}},\mathsf{p}) \in \mathrm{dom}(\Delta_2) \\ \Delta_1(\vec{\mathsf{y}},\mathsf{p}) & (\vec{\mathsf{y}},\mathsf{p}) \in \mathrm{dom}(\Delta_1) \setminus \mathrm{dom}(\Delta_2) \\ \text{undef} & \text{otherwise} \end{cases} \end{aligned}$$

The notion of mergeable pseudo-types is extended to specification pairs in order to type conditionals. Specifications Δ_1 and Δ_2 are *mergeable* when the local types they assign to sessions are mergeable. Formally, Δ_1 and Δ_2 are mergeable iff $\Delta_1|_{\mathbb{U}\cup\mathbb{Y}} = \Delta_2|_{\mathbb{U}\cup\mathbb{Y}}$, $\mathrm{dom}(\Delta_1) = \mathrm{dom}(\Delta_2)$ and, for all $(\vec{\mathsf{y}},\mathsf{p}) \in \mathrm{dom}(\Delta_1)$, $\Delta_1(\vec{\mathsf{y}},\mathsf{p})$ and $\Delta_2(\vec{\mathsf{y}},\mathsf{p})$ are mergeable. When Δ_1 and Δ_2 are mergeable, $\Delta_1 \bowtie \Delta_2$ merges the local types of sessions:

$$(\Delta_1 \bowtie \Delta_2)|_{\mathbb{U}\cup\mathbb{Y}} = \Delta_1|_{\mathbb{U}\cup\mathbb{Y}} \qquad \text{and} \qquad (\Delta_1 \bowtie \Delta_2)(\vec{\mathsf{y}},\mathsf{p}) = \Delta_1(\vec{\mathsf{y}},\mathsf{p}) \bowtie \Delta_2(\vec{\mathsf{y}},\mathsf{p})$$

We remark that the merge operation on specifications is idempotent, associative, and commutative. For conditionals we have

$$\text{[Vif]}\quad \frac{e \wedge e' \llcorner \Gamma \;\vdash\; P_1 \;\triangleright\; \Delta_1 \qquad e \wedge \neg e' \llcorner \Gamma \;\vdash\; P_2 \;\triangleright\; \Delta_2}{e \llcorner \Gamma \;\vdash\; \mathsf{if}\ e'\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2 \;\triangleright\; \Delta_1 \bowtie \Delta_2}$$

that requires to decompose the specification into two specifications Δ_1 and Δ_2 so to type conditional processes with the merge (*cf.* Definition 7.2) of Δ_1 and Δ_2. The premiss checks that the then-branch is typed in Δ_1 after extending the assumption e with the guard e' of the conditional while the else-branch is typed in Δ_2 after extending e with the negation of e'. Recall that judgements require consistency of their assumptions, hence rule [Vif] is not applicable if $e \wedge e'$ or $e \wedge \neg e'$ are inconsistent.

Example 7.3. Our typing distinguishes between B_1 and B_2 in Section 1 because B_1 is validated while B_2 is not. This is due to the rule [VIf]. In fact, after a few verification steps on B_1 we can apply rule [VIf] and prove the following judgement:

$$\mathtt{true} \lrcorner \Gamma \vdash \mathsf{if}\ (\mathit{check}\ c)\ \mathsf{then}\ \overline{\mathsf{ok}}\ \mathsf{else}\ \overline{\mathsf{ko}} \triangleright \Delta$$

where Γ assigns some sort to c and $\Delta = ((\mathsf{login}, \mathsf{deposit}, \mathsf{overdraft}, \mathsf{ok}, \mathsf{ko}), \mathsf{b}) : \overline{\mathsf{ok}} \oplus \overline{\mathsf{ko}}$. Instead, for B_2 we would have to prove

$$\mathtt{true} \lrcorner \Gamma \vdash \overline{\mathsf{ko}} \triangleright \Delta$$

which makes the validation of B_2 fail; this is due to the fact that the only rule for typing a sending process is [VSend], which cannot be applied against the specification Δ that assigns $\overline{\mathsf{ok}} \oplus \overline{\mathsf{ko}}$ to the participant's session. ◇

For-loops are typed with the following two rules

$$\text{[VForEnd]}\quad \frac{e \wedge \ell = \varepsilon \not\forall \bot \qquad \Gamma \vdash \ell\colon [\mathsf{d}] \qquad \Delta\ \mathsf{end\text{-}only}}{e \lrcorner \Gamma \vdash \mathsf{for}\ x\ \mathsf{in}\ \ell\ \mathsf{do}\ P \triangleright \Delta}$$

$$\text{[VFor]}\quad \frac{e \wedge \ell \neq \varepsilon \not\forall \bot \quad \Gamma \vdash \ell\colon [\mathsf{d}] \quad e \lrcorner \Gamma, x : \mathsf{d} \vdash P \triangleright \Delta \quad \Delta\ \text{active} \quad x \notin \mathrm{var}(\Delta)}{e \lrcorner \Gamma \vdash \mathsf{for}\ x\ \mathsf{in}\ \ell\ \mathsf{do}\ P \triangleright \Delta^\star}$$

Rule [VForEnd] handles the case in which the expression ℓ denotes an empty list, that is when the for-loop should be skipped. For this reason, the typing is similar to the typing of the idle process (rule [VEnd]). When the expression ℓ denotes a non-empty list under the assumption e (i.e., $\Gamma \vdash \ell\colon [\mathsf{d}]$ and $e \wedge \ell \neq \varepsilon \not\forall \bot$), we apply rule [VFor] to validate for-loops. The conclusion of the rule types the for-loop with $\Delta^\star$ which introduces iterative pseudo-types and is defined as follows:

$$\Delta^\star|_{\mathbb{U}\cup\mathbb{Y}} = \Delta|_{\mathbb{U}\cup\mathbb{Y}} \qquad \text{and} \qquad \Delta^\star(\vec{\mathsf{y}},\mathsf{p}) = (\Delta(\vec{\mathsf{y}},\mathsf{p}))^\star$$

Note that Δ has to type the body P under the context Γ extended with $x : \mathsf{d}$ because x can occur free in P. Moreover, P has to inform all other peers that the iteration continues. This is checked by the condition Δ *active*, namely that for all $(\vec{\mathsf{y}},\mathsf{p}) \in \mathrm{dom}(\Delta)$, $\mathsf{nf}(\Delta(\vec{\mathsf{y}},\mathsf{p})) = \bigoplus_{i\in I} e_i \prec \overline{\mathsf{y}_i}\ \mathsf{d}_i.\mathbb{T}_i$. Condition $x \notin \mathrm{var}(\Delta)$ ensures that guards of P do not depend on the iteration variable x, making each choice available at each iteration.

Rule [VLoop] below types passive processes of iterations.

$$\text{[VLoop]}\quad \frac{e \lrcorner \Gamma \vdash N \triangleright \Delta_1 \qquad e \lrcorner \Gamma \vdash M \triangleright \Delta_2 \qquad \Delta_1 \text{ and } \Delta_2 \text{ passively compatible}}{e \lrcorner \Gamma \vdash \mathsf{repeat}\ N\ \mathsf{until}\ M \triangleright \Delta_1^\star; \Delta_2}$$

The premiss of the rule types the iteration body N and the loop exit M with specifications Δ_1 and Δ_2 respectively. Both specifications are required to be *passively compatible*, i.e., there is just one session that decides whether to continue or terminate the iteration, and use different channels to communicate such choice. Formally, specifications $\Delta_1 = \Delta, (\vec{y},\mathsf{p}) : \mathbb{T}$ and $\Delta_2 = \Delta, (\vec{y},\mathsf{p}) : \mathbb{T}'$ are *passively compatible* iff

$$\mathrm{dom}(\Delta) \subseteq \mathbb{U} \cup \mathbb{Y}, \qquad \mathbb{T} = \sum_{i\in I} e_i \prec \mathsf{y}_i\ \mathsf{d}_i.\mathbb{T}_i \qquad \text{and} \qquad \mathbb{T}' = \sum_{j\in J} e_j \prec \mathsf{y}_j\ \mathsf{d}_j.\mathbb{T}_j$$

with $y_i \neq y_j$ for all $i \in I, j \in J$.

We now consider the typing rules for systems in Figure 11, which essentially deal with parallel composition, restriction of shared names, and queues. For parallel composition

$$\text{[VPar]}\quad \frac{e_1 \frown \Gamma \ \vdash\ S_1 \ \triangleright\ \Delta_1 \qquad e_2 \frown \Gamma \ \vdash\ S_2 \ \triangleright\ \Delta_2 \qquad \Delta_1 \text{ and } \Delta_2 \text{ independent}}{e_1 \wedge e_2 \frown \Gamma \ \vdash\ S_1 | S_2 \ \triangleright\ \Delta_1 \cup \Delta_2}$$

requires to split the specification into two *independent* specifications Δ_1 and Δ_2 that respectively type each side of the parallel. Specifications are independent when they agree on shared names and are disjoint on queues and participants' sessions; more precisely, Δ_1 and Δ_2 are *independent* when

- $\Delta_1|_{\mathbb{U}} = \Delta_2|_{\mathbb{U}}$ and $\mathrm{dom}(\Delta_1|_{\mathbb{Y}}) \cap \mathrm{dom}(\Delta_2|_{\mathbb{Y}}) = \emptyset$
- for all $(\vec{y},\mathsf{p}) \in \mathrm{dom}(\Delta_1)$ and $(\vec{y'},\mathsf{p}') \in \mathrm{dom}(\Delta_2)$, if $\vec{y} \cap \vec{y'} \neq \emptyset$ then $\vec{y} = \vec{y'}$ and $\mathsf{p} \neq \mathsf{p}'$

The union of independent specifications enjoys the sanity condition (7.2).

A restricted session is typed by

$$\text{[VNew]}\quad \frac{e \frown \Gamma \ \vdash\ S \ \triangleright\ \Delta}{e \frown \Gamma \ \vdash\ (\nu \vec{y}@u)S \ \triangleright\ \Delta|_{-\vec{y}}}$$

that removes participants' sessions and sessions' queues referring to the restricted names $\vec{y}$ from the specification Δ typing the scope S (this restricted specification is denoted as $\Delta|_{-\vec{y}}$).

The typing of queues is straightforwardly handled by the following two rules

$$\text{[VQueue]}\quad \frac{\vdash \mathsf{v} : \mathsf{d} \qquad e \frown \Gamma \ \vdash\ \mathsf{y}[\vec{\mathsf{v}}] \ \triangleright\ \mathsf{y} : [\vec{\mathsf{d}}]}{e \frown \Gamma \ \vdash\ \mathsf{y}[\vec{\mathsf{v}} \cdot \mathsf{v}] \ \triangleright\ \mathsf{y} : [\vec{\mathsf{d}} \cdot \mathsf{d}]} \qquad \text{[VEmpty]}\quad e \frown \Gamma \ \vdash\ \mathsf{y} : [] \ \triangleright\ \mathsf{y} : []$$

where $\_ \cdot \_$ denotes the concatenation operation on sequences and [VEmpty] permits to type empty queues.

7.3. **Typing the POP2 Protocol.** We now apply our type system to the implementations of our running example. We start by considering the process *Init* in 5.1 (page 16) and the specification $\Delta = u : \mathsf{G}_{\text{POP}}$ with G_{POP} from Section 3. We recall that its projection on s is $\mathsf{T}_\mathsf{s} = \mathsf{G}_{\text{POP}} \upharpoonright \mathsf{s}$ (also from 3). Then, the typing judgement for *Init* is obtained by using rule [VAcc] as follows

$$\text{[VAcc]}\quad \frac{\begin{array}{c}\vdots\\ \Delta(u) \equiv \mathsf{G}_{\text{POP}} \quad \mathtt{true} \frown \emptyset \ \vdash\ \mathit{Srv} \ \triangleright\ \Delta, (\vec{y},\mathsf{s}) : \mathbb{T}_\mathsf{s} \quad \eth(\mathbb{T}_\mathsf{s}) = \mathsf{T}_\mathsf{s}\end{array}}{\mathtt{true} \frown \emptyset \ \vdash\ u^{\mathsf{s}}(\vec{y}).\mathit{Srv} \ \triangleright\ \Delta}$$

[VReq]
$$\frac{\Delta(u) \equiv \mathcal{G}(\vec{\mathsf{y}}) \qquad e \llcorner \Gamma \vdash P \triangleright \Delta, (\vec{\mathsf{y}},0) : \mathbb{T} \qquad \jmath\big(\mathsf{nf}(\mathbb{T})\big) = \mathsf{nf}(\mathcal{G}(\vec{\mathsf{y}}) \upharpoonright 0)}{e \llcorner \Gamma \vdash \overline{u}^{\mathsf{n}}(\vec{\mathsf{y}}).P \triangleright \Delta}$$

[VAcc]
$$\frac{\Delta(u) \equiv \mathcal{G}(\vec{\mathsf{y}}) \qquad e \llcorner \Gamma \vdash P \triangleright \Delta, (\vec{\mathsf{y}},\mathsf{p}) : \mathbb{T} \qquad \jmath\big(\mathsf{nf}(\mathbb{T})\big) = \mathsf{nf}(\mathcal{G}(\vec{\mathsf{y}}) \upharpoonright \mathsf{p})}{e \llcorner \Gamma \vdash u^{\mathsf{p}}(\vec{\mathsf{y}}).P \triangleright \Delta}$$

[VRcv]
$$\frac{\forall i \in I : \mathsf{y}_i \in \vec{\mathsf{y}} \qquad e \llcorner \Gamma, x_i : \mathsf{d}_i \vdash P_i \triangleright \Delta, (\vec{\mathsf{y}},\mathsf{p}) : \mathbb{T}_i}{e \llcorner \Gamma \vdash \sum_{i \in I} \mathsf{y}_i(x_i).P_i \triangleright \Delta, (\vec{\mathsf{y}},\mathsf{p}) : \sum_{i \in I} e \prec \mathsf{y}_i\ \mathsf{d}_i.\mathbb{T}_i}$$

[VSend]
$$\frac{\Gamma \vdash e' : \mathsf{d} \qquad \mathsf{y} \in \vec{\mathsf{y}} \qquad \Delta \text{ end-only}}{e \llcorner \Gamma \vdash \overline{\mathsf{y}}\, e' \triangleright \Delta, (\vec{\mathsf{y}},\mathsf{p}) : e \prec \overline{\mathsf{y}}\ \mathsf{d}; e \prec \mathsf{end}}$$

[VEnd]
$$\frac{\Delta \text{ end-only}}{e \llcorner \Gamma \vdash 0 \triangleright \Delta}$$

[VSeq]
$$\frac{e \llcorner \Gamma \vdash P_1 \triangleright \Delta_1 \qquad e \llcorner \Gamma \vdash P_2 \triangleright \Delta_2}{e \llcorner \Gamma \vdash P_1; P_2 \triangleright \Delta_1; \Delta_2}$$

[VIf]
$$\frac{e \wedge e' \llcorner \Gamma \vdash P_1 \triangleright \Delta_1 \qquad e \wedge \neg e' \llcorner \Gamma \vdash P_2 \triangleright \Delta_2}{e \llcorner \Gamma \vdash \mathsf{if}\ e'\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2 \triangleright \Delta_1 \bowtie \Delta_2}$$

[VForEnd]
$$\frac{e \wedge \ell = \varepsilon \not\models \bot \qquad \Gamma \vdash \ell : [\mathsf{d}] \qquad \Delta \text{ end-only}}{e \llcorner \Gamma \vdash \mathsf{for}\ x\ \mathsf{in}\ \ell\ \mathsf{do}\ P \triangleright \Delta}$$

[VFor]
$$\frac{e \wedge \ell \neq \varepsilon \not\models \bot \quad \Gamma \vdash \ell : [\mathsf{d}] \quad e \llcorner \Gamma, x : \mathsf{d} \vdash P \triangleright \Delta \quad \Delta \text{ active} \quad x \notin \mathrm{var}(\Delta)}{e \llcorner \Gamma \vdash \mathsf{for}\ x\ \mathsf{in}\ \ell\ \mathsf{do}\ P \triangleright \Delta^{\star}}$$

[VLoop]
$$\frac{e \llcorner \Gamma \vdash N \triangleright \Delta_1 \qquad e \llcorner \Gamma \vdash M \triangleright \Delta_2 \qquad \Delta_1 \text{ and } \Delta_2 \text{ passively compatible}}{e \llcorner \Gamma \vdash \mathsf{repeat}\ N\ \mathsf{until}\ M \triangleright \Delta_1^{\star}; \Delta_2}$$

Figure 10: Typing rules for processes

where the continuation Srv is typed against the specification Δ extended with a new participant's session $(\vec{\mathsf{y}},\mathsf{s})$ whose type $\mathbb{T}_{\mathsf{s}}$ matches T_{s}, i.e., $\jmath\big(\mathbb{T}_{\mathsf{s}}\big) = \mathsf{T}_{\mathsf{s}}$. Such $\mathbb{T}_{\mathsf{s}}$ is obtained from the judgement $\mathtt{true} \llcorner \emptyset \vdash \mathit{Srv} \triangleright \Delta, (\vec{\mathsf{y}},\mathsf{s}) : \mathbb{T}_{\mathsf{s}}$. Since $\mathit{Srv} \triangleq \mathsf{quit}.\mathit{Exit} + \mathsf{helo}(c).\mathit{Mbox}(c)$

$$\text{[VPar]}\quad \frac{e_1 \frown \Gamma \vdash S_1 \triangleright \Delta_1 \qquad e_2 \frown \Gamma \vdash S_2 \triangleright \Delta_2 \qquad \Delta_1 \text{ and } \Delta_2 \text{ independent}}{e_1 \wedge e_2 \frown \Gamma \vdash S_1 | S_2 \triangleright \Delta_1 \cup \Delta_2}$$

$$\text{[VQueue]}\quad \frac{\vdash \mathsf{v} : \mathsf{d} \qquad e \frown \Gamma \vdash \mathsf{y}[\vec{\mathsf{v}}] \triangleright \mathsf{y} : [\vec{\mathsf{d}}]}{e \frown \Gamma \vdash \mathsf{y}[\vec{\mathsf{v}} \cdot \mathsf{v}] \triangleright \mathsf{y} : [\vec{\mathsf{d}} \cdot \mathsf{d}]}$$

$$\text{[VEmpty]}\quad e \frown \Gamma \vdash \mathsf{y} : [] \triangleright \mathsf{y} : []$$

$$\text{[VNew]}\quad \frac{e \frown \Gamma \vdash S \triangleright \Delta}{e \frown \Gamma \vdash (\nu \vec{\mathsf{y}} @ u) S \triangleright \Delta|_{-\vec{\mathsf{y}}}}$$

Figure 11: Typing rules for systems

is an input-guarded process, the judgement is obtained by applying rule [VRcv] as follows:

$$\frac{\vdots \qquad\qquad \vdots}{\mathtt{true} \frown \emptyset \vdash \mathit{Exit} \triangleright \Delta, (\vec{\mathsf{y}},\mathsf{s}) : \mathbb{T}_{\text{EXIT}} \qquad \mathtt{true} \frown c : \mathsf{Str} \vdash \mathit{Mbox}(c) \triangleright \Delta, (\vec{\mathsf{y}},\mathsf{s}) : \mathbb{T}_{\text{MBOX}}}{\mathtt{true} \frown \emptyset \vdash \mathsf{quit}.\mathit{Exit} + \mathsf{helo}(c).\mathit{Mbox}(c) \triangleright \Delta, (\vec{\mathsf{y}},\mathsf{s}) : \mathbb{T}_{\mathsf{s}}}\ \text{[VRcv]} \tag{7.3}$$

where

$$\mathbb{T}_{\mathsf{s}} \triangleq \mathtt{true} \prec \mathsf{quit}\,.\mathbb{T}_{\text{EXIT}} \;+\; \mathtt{true} \prec \mathsf{helo}\ \mathsf{Str}.\mathbb{T}_{\text{MBOX}}$$

with $\partial(\mathbb{T}_{\text{EXIT}}) = \mathsf{T}_{\text{EXIT}}$ and $\partial(\mathbb{T}_{\text{MBOX}}) = \mathsf{T}_{\text{MBOX}}$ so to satisfy $\partial(\mathbb{T}_{\mathsf{s}}) = \mathsf{T}_{\mathsf{s}}$. The first premiss in (7.3) is derived as follows by taking $\mathbb{T}_{\text{EXIT}} = \mathtt{true} \prec \overline{\mathsf{bye}}\,.\mathtt{true} \prec \mathsf{end}$ (recall that $\mathit{Exit} \triangleq \overline{\mathsf{bye}}$ and $\overline{\mathsf{bye}}$ is a shorthand for $\overline{\mathsf{bye}}\,()$).

$$\frac{\emptyset \vdash () : \mathsf{Unit} \qquad \mathsf{bye} \in \vec{\mathsf{y}} \qquad (\Delta, (\vec{\mathsf{y}},\mathsf{s}) : \mathtt{true} \prec \mathsf{end}) \text{ end-only}}{\mathtt{true} \frown \emptyset \vdash \overline{\mathsf{bye}}\,() \triangleright \Delta, (\vec{\mathsf{y}},\mathsf{s}) : \mathbb{T}_{\text{EXIT}}}\ \text{[VSend]}$$

The second premiss in (7.3) follows by using rule [VIf] because $\mathit{Mbox}(c)$ in (5.3) is a conditional process (hereafter, we write e as shorthand for $\mathsf{auth}\ c$).

$$\frac{\vdots \qquad\qquad \vdots}{e \frown c : \mathsf{Str} \vdash \overline{\mathsf{r}}\,(\mathsf{mn}\ \mathsf{inbox}); \mathit{Nmbr} \triangleright \Delta, (\vec{\mathsf{y}},\mathsf{s}) : \mathbb{T}_{\mathsf{then}} \qquad \neg e \frown c : \mathsf{Str} \vdash \overline{\mathsf{e}}\,; \mathit{Exit} \triangleright \Delta, (\vec{\mathsf{y}},\mathsf{s}) : \mathbb{T}_{\mathsf{else}}}{\mathtt{true} \frown c : \mathsf{Str} \vdash \mathit{Mbox}(c) \triangleright \Delta, (\vec{\mathsf{y}},\mathsf{s}) : \mathbb{T}_{\text{MBOX}}}\ \text{[VIf]} \tag{7.4}$$

with $\mathbb{T}_{\text{MBOX}} = \mathbb{T}_{\mathsf{then}} \bowtie \mathbb{T}_{\mathsf{else}}$. The second premiss above can be shown by using the rules [VSeq] and [VSend] and by taking

$$\mathbb{T}_{\mathsf{else}} = \neg e \prec \overline{\mathsf{e}}\,; \neg e \prec \mathsf{end}; \neg e \prec \overline{\mathsf{bye}}\,; \neg e \prec \mathsf{end}$$

For the first premiss in (7.4) we use rule [Seq] as follows

$$\frac{\vdots \qquad\qquad \vdots}{e \frown c : \mathsf{Str} \vdash \overline{\mathsf{r}}\,(\mathsf{mn}\ \mathsf{inbox}) \triangleright \Delta, (\vec{\mathsf{y}},\mathsf{s}) : \mathbb{T}_{\mathsf{then}_1} \qquad e \frown c : \mathsf{Str} \vdash \mathit{Nmbr} \triangleright \Delta, (\vec{\mathsf{y}},\mathsf{s}) : \mathbb{T}_{\mathsf{then}_2}}{e \frown c : \mathsf{Str} \vdash \overline{\mathsf{r}}\,(\mathsf{mn}\ \mathsf{inbox}); \mathit{Nmbr} \triangleright \Delta, (\vec{\mathsf{y}},\mathsf{s}) : \mathbb{T}_{\mathsf{then}}}\ \text{[VSeq]} \tag{7.5}$$

with $\mathbb{T}_{\mathtt{then}} = \mathbb{T}_{\mathtt{then}_1}; \mathbb{T}_{\mathtt{then}_2}$. By applying [VSend], we conclude that $\mathbb{T}_{\mathtt{then}_1} = e \prec \overline{\mathsf{r}}\ \mathsf{Int}; e \prec \mathsf{end}$. Since *Nmbr* is a repeat-until loop, the second premiss in (7.5) is obtained by using rule [Vloop] (we write $Nmbr_{body}$ for the body of the iteration and $Nmbr_{until}$ for the until guard).

$$\dfrac{\dfrac{\vdots}{e \lrcorner c : \mathsf{Str} \vdash Nmbr_{body} \triangleright \Delta, (\vec{y},\mathsf{s}) : \mathbb{T}_{\mathtt{body}}} \qquad \dfrac{\vdots}{e \lrcorner c : \mathsf{Str} \vdash Nmbr_{until} \triangleright \Delta, (\vec{y},\mathsf{s}) : \mathbb{T}_{\mathtt{until}}}}{e \lrcorner c : \mathsf{Str} \vdash Nmbr \triangleright \Delta, (\vec{y},\mathsf{s}) : \mathbb{T}_{\mathtt{then}_2}}\ [\mathsf{VLoop}]$$

with $\Delta, (\vec{y},\mathsf{s}) : \mathbb{T}_{\mathtt{body}}$ and $\Delta, (\vec{y},\mathsf{s}) : \mathbb{T}_{\mathtt{until}}$ passively compatible and $\mathbb{T}_{\mathtt{t}}hen_2 = (\mathbb{T}_{\mathtt{body}})^{\star}; \mathbb{T}_{\mathtt{until}}$. By following the same approach, it can be shown that both premisses are derivable by taking

$$\begin{array}{rcl} \mathbb{T}_{\mathtt{body}} & = & e \prec \mathsf{fold}\ \mathsf{Str}.e \prec \overline{\mathsf{r}}\ \mathsf{Int}; e \prec \mathsf{end} + e \prec \mathsf{read}\ \mathsf{Int}.e \prec \overline{\mathsf{r}}\ \mathsf{Int}; \mathbb{T}_{\mathtt{SIZE}} \\ \mathbb{T}_{\mathtt{until}} & = & e \prec \mathsf{quit}\ .e \prec \overline{\mathsf{bye}}\ ; e \prec \mathsf{end} \end{array}$$

for a suitable $\mathbb{T}_{\mathtt{SIZE}}$ such that $\jmath(\mathbb{T}_{\mathtt{SIZE}}) = \mathsf{T}_{\mathtt{SIZE}}$. It is straightforward to check that $\Delta, (\vec{y},\mathsf{s}) : \mathbb{T}_{\mathtt{body}}$ and $\Delta, (\vec{y},\mathsf{s}) : \mathbb{T}_{\mathtt{until}}$ are passively compatible because the channels in $\mathbb{T}_{\mathtt{body}}$ are different from the ones appearing in $\mathbb{T}_{\mathtt{until}}$. Moreover, $\jmath(\mathbb{T}_{\mathtt{then}_2}) = \jmath((\mathbb{T}_{\mathtt{body}})^{\star}; \mathbb{T}_{\mathtt{until}}) = \mathsf{T}_{\mathtt{Nmbr}}$.

It remains to show that $\mathbb{T}_{\mathtt{MBOX}} = \mathbb{T}_{\mathtt{then}} \bowtie \mathbb{T}_{\mathtt{else}}$ is well-defined. We first compute the normal form of the pseudo types.

$$\begin{array}{rcl} \mathsf{nf}(\mathbb{T}_{\mathtt{then}}) & = & e \prec \overline{\mathsf{r}}\ \mathsf{Int}; \mathsf{nf}(\mathbb{T}_{\mathtt{then}_2}) \\ \mathsf{nf}(\mathbb{T}_{\mathtt{else}}) & = & \neg e \prec \overline{\mathsf{e}}\ ; \neg e \prec \overline{\mathsf{bye}}\ ; \neg e \prec \mathsf{end} \end{array}$$

It is immediate to notice that $\mathsf{nf}(\mathbb{T}_{\mathtt{then}})$ and $\mathsf{nf}(\mathbb{T}_{\mathtt{else}})$ are mergeable because they are internal choices on disjoint set of session channels. Therefore,

$$\mathbb{T}_{\mathtt{MBOX}} = \mathbb{T}_{\mathtt{then}} \bowtie \mathbb{T}_{\mathtt{else}} = e \prec \overline{\mathsf{r}}\ \mathsf{Int}; \mathsf{nf}(\mathbb{T}_{\mathtt{then}_2}) \ \oplus\ \neg e \prec \overline{\mathsf{e}}\ ; \neg e \prec \overline{\mathsf{bye}}\ ; \neg e \prec \mathsf{end}$$

Finally, note that $\jmath(\mathbb{T}_{\mathtt{MBOX}}) = \mathsf{T}_{\mathtt{MBOX}}$.

We now give the main types for the multiparty variant given in (5.5) (*cf.* page 17). Assume $\Delta(u) \equiv \mathsf{G}'_{\mathtt{POP}}$ from Section 3 and consider the following pseudo-type:

$$\begin{array}{rcl} \mathbb{T}'_{\mathtt{s}} & \triangleq & \mathtt{true} \prec \mathsf{quit}\ .\mathbb{T}_{\mathtt{EXIT}}\ +\ \mathtt{true} \prec \mathsf{helo}\ \mathsf{Str}.\mathbb{T}_{\mathtt{AUTH}} \\ \mathbb{T}_{\mathtt{AUTH}} & \triangleq & \mathtt{true} \prec \overline{\mathsf{req}}\ \mathsf{Str}; \mathtt{true} \prec \mathsf{res}\ \mathsf{Bool}.\mathbb{T}'_{\mathtt{MBOX}} \\ \mathbb{T}'_{\mathtt{MBOX}} & \triangleq & e \wedge a \prec \overline{\mathsf{r}}\ \mathsf{Int}; \mathbb{T}'_{\mathtt{NMBR}}\ \oplus\ \neg(e \wedge a) \prec \overline{\mathsf{e}}\ ; \mathbb{T}'_{\mathtt{EXIT}} \end{array}$$

such that $\jmath(\mathbb{T}'_{\mathsf{s}}) = \mathsf{T}'_{\mathsf{s}}$, $\mathbb{T}_{\mathtt{NMBR}}$ is as $\mathbb{T}_{\mathtt{then}_2}$ above except that all enabling conditions are $(e \wedge a)$, and $\mathbb{T}'_{\mathtt{EXIT}}$ is as $\mathbb{T}_{\mathtt{EXIT}}$ except that all enabling conditions are $\neg(e \wedge a)$. The typing judgement $\mathtt{true} \lrcorner \emptyset \vdash Init' \triangleright \Delta, (\vec{y},\mathsf{s}) : \mathbb{T}'_{\mathtt{s}}$ can be obtained as in the previous case.

8. Properties of the Type System

In this section we show that a well-typed process (i) behaves as specified by the global type (Theorem 8.4) and (ii) is a WSI of the role played in the global type (Theorem 8.8).

$$[\mathsf{TReq}]\ \frac{\Delta(u) \equiv \mathcal{G}(\vec{\mathsf{y}}) \quad \eth(\mathbb{T}) = \mathsf{nf}(\mathcal{G}(\vec{\mathsf{y}})\upharpoonright 0)}{\Delta \xrightarrow{\overline{u}^{\mathsf{n}}(\vec{\mathsf{y}})} \Delta, (\vec{\mathsf{y}},0) : \mathbb{T}} \qquad [\mathsf{TRcv}]\ \frac{j \in I}{\Delta, (\vec{\mathsf{y}},\mathsf{p}) : \sum_{i\in I} e_i \prec \mathsf{y}_i\ \mathsf{d}_i.\mathbb{T}_i \xrightarrow{\mathsf{y}_j\mathsf{d}_j} \Delta, (\vec{\mathsf{y}},\mathsf{p}) : \mathbb{T}_j}$$

$$[\mathsf{TAcc}]\ \frac{\Delta(u) \equiv \mathcal{G}(\vec{\mathsf{y}}) \quad \eth(\mathbb{T}) = \mathsf{nf}(\mathcal{G}(\vec{\mathsf{y}})\upharpoonright \mathsf{p})}{\Delta \xrightarrow{u^{\mathsf{p}}(\vec{\mathsf{y}})} \Delta, (\vec{\mathsf{y}},\mathsf{p}) : \mathbb{T}} \qquad [\mathsf{TSend}]\ \frac{j \in I}{\Delta, (\vec{\mathsf{y}},\mathsf{p}) : \bigoplus_{i\in I} e_i \prec \overline{\mathsf{y}_i}\ \mathsf{d}_i.\mathbb{T}_i \xrightarrow{\overline{\mathsf{y}_j}\mathsf{d}_j} \Delta, (\vec{\mathsf{y}},\mathsf{p}) : \mathbb{T}_j}$$

$$[\mathsf{TSeq}]\ \frac{\Delta_1 \xrightarrow{\alpha} \Delta_1'}{\Delta_1;\Delta_2 \xrightarrow{\alpha} \Delta_1';\Delta_2} \qquad [\mathsf{TLoop}_0]\ \frac{\Delta_2 \xrightarrow{\mathsf{yd}} \Delta_2'}{\Delta_1^\star;\Delta_2 \xrightarrow{\mathsf{yd}} \Delta_2'} \qquad [\mathsf{TLoop}_1]\ \frac{\Delta \xrightarrow{\alpha} \Delta'}{\Delta^\star \xrightarrow{\alpha} \Delta'} \qquad [\mathsf{TLoop}_2]\ \frac{\Delta \xrightarrow{\alpha} \Delta'}{\Delta^\star \xrightarrow{\alpha} \Delta';\Delta^\star}$$

$$[\mathsf{TInit}]\ \frac{\Delta(u) \equiv \mathcal{G}(\vec{\mathsf{y}}) \triangleq \mathsf{G} \qquad \mathcal{P}(\mathsf{G}) = \{\mathsf{p}_0,\dots,\mathsf{p}_n\} \qquad \eth(\mathbb{T}_i) = \mathsf{nf}(\mathsf{G}\upharpoonright \mathsf{p}_i)\ \forall i \in \{0,\dots,n\}}{\Delta \xrightarrow{\tau} \Delta, (\vec{\mathsf{y}},\mathsf{p}_0) : \mathbb{T}_0,\dots,(\vec{\mathsf{y}},\mathsf{p}_n) : \mathbb{T}_n, \vec{\mathsf{y}} : []}$$

$$[\mathsf{TCom}_1]\ \frac{j \in I}{\Delta, (\vec{\mathsf{y}},\mathsf{p}) : \bigoplus_{i\in I} e_i \prec \overline{\mathsf{y}_i}\ \mathsf{d}_i.\mathbb{T}_i, \mathsf{y}_j : [\vec{\mathsf{d}}] \xrightarrow{\tau} \Delta, (\vec{\mathsf{y}},\mathsf{p}) : \mathbb{T}_j, \mathsf{y}_j : [\vec{\mathsf{d}} \cdot \mathsf{d}_j]}$$

$$[\mathsf{TCom}_2]\ \frac{j \in I}{\Delta, (\vec{\mathsf{y}},\mathsf{p}) : \sum_{i\in I} e_i \prec \mathsf{y}_i\ \mathsf{d}_i.\mathbb{T}_i, \mathsf{y}_j : [\mathsf{d}_j \cdot \vec{\mathsf{d}}] \xrightarrow{\tau} \Delta, (\vec{\mathsf{y}},\mathsf{p}) : \mathbb{T}_j, \mathsf{y}_j : [\vec{\mathsf{d}}]}$$

Figure 12: Labelled transitions for specifications

8.1. **Conformance.** In order to show that any well-typed process adheres to the behaviour defined by a global type, we relate the semantics of the process with the one of its specification through a subject reduction result (Theorem 8.2). The operational semantics of specifications is generated by the rules in Figure 12, where it is implicitly assumed that we work up-to normal forms, namely the pseudo-types are normalised before and after transitions. Notice that the labels are as in (4.1) on page 14 but for the fact that they cannot be conditional actions $e \vdash \alpha$. Intuitively, the rules in Figure 12, barred the last three, state how the specification of a single participant behaves in a session $\vec{\mathsf{y}}$ and are instrumental to establish subject reduction.

Rules [TReq] and [TAcc] account for a specification that initiates a new session by projecting (on 0 and p, resp.) the global type associated with the shared name u in $\mathrm{dom}(\Delta)$. Note that $\mathbb{T}$ can use arbitrary guards in the projections $\mathcal{G}(\vec{\mathsf{y}})\upharpoonright \mathsf{p}$ as long as the normal form of $\mathbb{T}$ matches the one of the projection. Rule [TRcv] accounts for the reception of a message. Dually, rule [TSend] accounts for an endpoint that performs one of its outputs. Rule [TSeq] relies on the definition of sequential composition of specifications (*cf.* page 26); observe that the case in which all pseudo-types in Δ_1 are of the form $e \prec \mathsf{end}$ is precluded because we work up-to normal form of pseudo-types. Finally, an iterative local type can be skipped (rule [TLoop_0]), executed once (rule [TLoop_1]), or be unfolded (rule [TLoop_2]). The last three rules in Figure 12 state how specifications of systems behave. Rule [TInit] initiates a new session $\vec{\mathsf{y}}$

$$[\mathsf{CEnd}]\ \frac{e\&\sigma}{\sigma \ltimes e \prec \mathsf{end}} \qquad [\mathsf{CSeq}]\ \frac{\sigma \ltimes \mathbb{T}_1 \qquad \sigma \ltimes \mathbb{T}_2}{\sigma \ltimes \mathbb{T}_1;\mathbb{T}_2} \qquad [\mathsf{CLoop}]\ \frac{\sigma \ltimes \mathbb{T}}{\sigma \ltimes \mathbb{T}^\star}$$

$$[\mathsf{CSend}]\ \frac{\exists i \in I\ :\ e_i\&\sigma \wedge \sigma \ltimes \mathbb{T}_i}{\sigma \ltimes \bigoplus_{i\in I} e_i \prec \overline{\mathsf{y}_i}\ \mathsf{d}_i.\mathbb{T}_i} \qquad [\mathsf{CRcv}]\ \frac{\forall i \in I\ :\ e_i\&\sigma \wedge \sigma \ltimes \mathbb{T}_i}{\sigma \ltimes \sum_{i\in I} e_i \prec \mathsf{y}_i\ \mathsf{d}_i.\mathbb{T}_i}$$

Figure 13: Consistency relation between stores and pseudo-types

by assigning each participant with a type that matches the corresponding projection of the global type. Rules [TCom_1] and [TCom_2] establish how specifications send and receive messages through queues.

The behaviour of processes depends on the stores they run on. Consequently, we compare process and specifications with respect to stores; concretely, we only consider the behaviour of processes running on stores that are consistent with the guards in the pseudo-types of the specifications. The consistency predicate $\_ \ltimes \_$ relates stores with pseudo-types and is inductively defined by the rules in Figure 13. Intuitively, $\sigma \ltimes \mathbb{T}$ holds if σ does not falsify any guard in $\mathbb{T}$; which is checked by rules [CEnd], [CSend] and [CRcv], where $e\&\sigma$ means $e \downarrow \sigma = \texttt{true}$ or $e \downarrow \sigma$ undefined.

The notion of consistency is then extended to type judgments as follows.

Definition 8.1 (Consistency)**.** A store σ is *consistent* with a judgement $e \llcorner \Gamma \vdash S \triangleright \Delta$, written $\sigma \ltimes (e;\Gamma;S;\Delta)$, if

(1) $\mathrm{dom}(\Delta|_{\mathbb{Y}}) \cup \mathrm{dom}(\Gamma) \subseteq \mathrm{dom}(\sigma)$
(2) $\forall x \in \mathrm{dom}(\Gamma)\ :\ \ \vdash \sigma(x) : \Gamma(x)$
(3) $e \downarrow \sigma = \texttt{true}$
(4) $\forall (\vec{\mathsf{y}},\mathsf{p}) \in \mathrm{dom}(\Delta)\ :\ \sigma \ltimes \Delta(\vec{\mathsf{y}},\mathsf{p})$.

A store σ is consistent with a type judgement $e \llcorner \Gamma \vdash S \triangleright \Delta$ if σ contains an assignment for any free name of S (Item 1); the values assigned to variables should match the type assigned by the environment Γ (Item 2). Besides, the typing assumption e and the guards in the pseudo-types in Δ should hold when evaluated over σ (Item 3 and Item 4).

Theorem 8.2 (Subject reduction)**.** *If* $e \llcorner \Gamma \vdash S \triangleright \Delta$, $\sigma \ltimes (e;\Gamma;S;\Delta)$, *and* $\langle S,\sigma\rangle \xrightarrow{e'\vdash\alpha} \langle S',\sigma'\rangle$ *then there exist* Γ' *and* Δ' *such that*

(1) *if* $\alpha = \mathsf{y}\mathsf{v}$ *then* $\Delta \xrightarrow{\mathsf{y}\mathsf{d}} \Delta'$ *for a sort* d*; moreover, if* $\vdash \mathsf{v} : \mathsf{d}$ *then there is* $x \in \mathbb{X}$ *such that*
$$e \wedge e' \llcorner \Gamma', x : \mathsf{d} \vdash S' \triangleright \Delta', \qquad \sigma'(x) = \mathsf{v}, \quad \text{and} \quad \sigma' \ltimes (e \wedge e';\Gamma', x : \mathsf{d};S';\Delta')$$
(2) *if* $\alpha = \overline{\mathsf{y}}\mathsf{v}$ *then* $\Delta \xrightarrow{\overline{\mathsf{y}}\mathsf{d}} \Delta'$ *with* $\vdash \mathsf{v} : \mathsf{d}$, $e \wedge e' \llcorner \Gamma' \vdash S' \triangleright \Delta'$, *and* $\sigma' \ltimes (e \wedge e';\Gamma';S';\Delta')$
(3) *otherwise* $\Delta \xrightarrow{\alpha} \Delta'$, *and* $e \wedge e' \llcorner \Gamma' \vdash S' \triangleright \Delta'$, $\sigma' \ltimes (e \wedge e';\Gamma';S';\Delta')$.

The typing rules in Section 7 ensure the semantic conformance of processes with the behaviour prescribed by their types. Here, we define conformance in terms of *conditional that* simulation relates states and specifications. Our definition is standard, except for input actions, for which specifications have to simulate only inputs of messages with the expected type (i.e., systems are not responsible when receiving ill-typed messages).

Define $\stackrel{\alpha}{\Longrightarrow} = \stackrel{\tau}{\rightarrow}^* \stackrel{\alpha}{\rightarrow}$. Let $\Delta \stackrel{\mathsf{y}}{\Longrightarrow}$ mean that there are Δ' and v such that $\Delta \stackrel{\mathsf{yv}}{\Longrightarrow} \Delta'$.

Definition 8.3 (Conditional simulation)**.** A relation $\mathbb{R}$ between pairs state-specification is a *conditional simulation* if for any $(\langle S, \sigma\rangle, \Delta) \in \mathbb{R}$, if $\langle S, \sigma\rangle \xrightarrow{e \vdash \alpha} \langle S', \sigma'\rangle$ then

(1) if $\alpha = \mathsf{yv}$ then there exists Δ' such that $\Delta \stackrel{\mathsf{yd}}{\Longrightarrow} \Delta'$ and if $\vdash \mathsf{v} : \mathsf{d}$ then there exists x such that $\sigma' = \sigma[x \mapsto \mathsf{v}]$ and $(\langle S', \sigma'\rangle, \Delta') \in \mathbb{R}$
(2) otherwise, $\Delta \stackrel{\alpha}{\Longrightarrow} \Delta'$ and $(\langle S', \sigma'\rangle, \Delta') \in \mathbb{R}$.

We write $\langle S, \sigma\rangle \precsim \Delta$ if there is a conditional simulation $\mathbb{R}$ such that $(\langle S, \sigma\rangle, \Delta) \in \mathbb{R}$.

By (1), only inputs of S with the expected type have to be matched by Δ (recall rule [TRec] in Figure 12), while it is no longer expected to conform to the specification after an ill-typed input (i.e., not allowed by Δ).

Conformance follows by straightforward coinduction from Theorem 8.2.

Theorem 8.4 (Conformance)**.** *If* $e \lrcorner \Gamma \vdash S \triangleright \Delta$ *and* $\sigma \ltimes (e; \Gamma; S; \Delta)$ *then* $\langle S, \sigma\rangle \precsim \Delta$.

Proof. Using Theorem 8.2 it is straightforward to show that

$$\mathbb{R} = \{(\langle S, \sigma\rangle, \Delta) \mid e \lrcorner \Gamma \vdash S \triangleright \Delta \text{ and } \sigma \ltimes (e; \Gamma; S; \Delta)\}$$

is a conditional simulation. □

8.2. **WSI by Typing.** We show that well-typed processes are WSIs (Definition 6.8 on page 22). First, we relate the runs of a global type with those of its corresponding specifications. Then, we state the correspondence between the runs of specifications and well-typed implementations. A set of implementations covering a global type $\mathcal{G}$ can exhibit more behaviour than the runs of $\mathcal{G}$. Nonetheless, we use WSI with our subject reduction property (*cf.* Definition 8.3 and Theorem 8.4) to characterise valid implementations.

Given a specification Δ such that $\Delta(\vec{\mathsf{y}},\mathsf{p})$ is in normal form for all $(\vec{\mathsf{y}},\mathsf{p}) \in \mathrm{dom}(\Delta)$, we let $\mathcal{R}_{\vec{\mathsf{y}}}(\Delta)$, inductively defined by the rules in Figure 14, to be of the set of runs of session $\vec{\mathsf{y}}$ generated by Δ. Rule [RTCom$_1$] accounts for runs starting with output actions $\langle \mathsf{p}, \overline{\mathsf{y}_j}\mathsf{d}_j\rangle$ performed by an endpoint $(\vec{\mathsf{y}},\mathsf{p})$ and followed by a run r of the continuation. Rule [RTCom$_2$] analogously deals with inputs. Rules [RTIt$_1$] and [RTIt$_2$] unfold an iterative type; one iteration is mandatory while the additional ones are optional. The remaining rules are self-explanatory. Since all types in Δ are in normal form (and therefore inconsistent guards may appear only in sub terms $e \prec \mathsf{end}$), the rules in Figure 14 do not generate runs with actions that cannot be fired. The following two results establish the correspondence between the denotational and operational semantics of specifications.

Lemma 8.5. *Let* Δ *be a specification such that for all* $(\vec{\mathsf{y}},\mathsf{p}) \in \mathrm{dom}(\Delta)$, $\Delta(\vec{\mathsf{y}},\mathsf{p})$ *is in normal form. If* $\Delta \stackrel{\tau}{\rightarrow} \Delta'$, *then for all* $r \in \mathcal{R}_{\vec{\mathsf{y}}}(\Delta')$ *either:*

- $r \in \mathcal{R}_{\vec{\mathsf{y}}}(\Delta)$, *or*
- $\langle \mathsf{p}, \overline{\mathsf{y}}\mathsf{d}\rangle r \in \mathcal{R}_{\vec{\mathsf{y}}}(\Delta)$, *or else*
- $\langle \mathsf{q}, \mathsf{y}\ \mathsf{d}\rangle r \in \mathcal{R}_{\vec{\mathsf{y}}}(\Delta)$.

Lemma 8.6. *Let* Δ *a specification such that for all* $(\vec{\mathsf{y}},\mathsf{p}) \in \mathrm{dom}(\Delta)$, $\Delta(\vec{\mathsf{y}},\mathsf{p})$ *is in normal form. If* $r \in \mathcal{R}_{\vec{\mathsf{y}}}(\Delta)$ *and* $r \neq \epsilon$ *then* $\Delta \stackrel{\tau}{\rightarrow} \Delta'$ *and either*

- $r \in \mathcal{R}_{\vec{\mathsf{y}}}(\Delta')$, *or*
- $r = \langle \mathsf{p}, \overline{\mathsf{y}}\mathsf{d}\rangle r'$ *and* $r' \in \mathcal{R}_{\vec{\mathsf{y}}}(\Delta')$, *or*

$$[\mathsf{RTCom}_1]\quad \frac{j \in I \qquad r \in \mathcal{R}_{\vec{y}}(\Delta, (\vec{y},\mathsf{p}) : \mathbb{T}_j, \mathsf{y}_j : [\vec{\mathsf{d}} \cdot \mathsf{d}_j])}{\langle \mathsf{p}, \overline{\mathsf{y}_j}\mathsf{d}_j\rangle r \in \mathcal{R}_{\vec{y}}(\Delta, (\vec{y},\mathsf{p}) : \bigoplus_{i\in I} e_i \prec \overline{\mathsf{y}_i}\ \mathsf{d}_i.\mathbb{T}_i, \mathsf{y}_j : [\vec{\mathsf{d}}])}$$

$$[\mathsf{RTCom}_2]\quad \frac{j \in I \qquad r \in \mathcal{R}_{\vec{y}}(\Delta, (\vec{y},\mathsf{p}) : \mathbb{T}_j, \mathsf{y}_j : [\vec{\mathsf{d}}])}{\langle \mathsf{p}, \mathsf{y}_j\mathsf{d}_j\rangle r \in \mathcal{R}_{\vec{y}}(\Delta, (\vec{y},\mathsf{p}) : \sum_{i\in I} e_i \prec \mathsf{y}_i\ \mathsf{d}_i.\mathbb{T}_i, \mathsf{y}_j : [\mathsf{d}_j \cdot \vec{\mathsf{d}}])}$$

$$[\mathsf{RTIt}_1]\quad \frac{r \in \mathcal{R}_{\vec{y}}(\Delta)}{r \in \mathcal{R}_{\vec{y}}(\Delta^\star)} \qquad [\mathsf{RTIt}_2]\quad \frac{r_1 \in \mathcal{R}_{\vec{y}}(\Delta_1) \qquad r_2 \in \mathcal{R}_{\vec{y}}(\Delta_1^\star)}{r_1[r_2] \in \mathcal{R}_{\vec{y}}(\Delta_1^\star)}$$

$$[\mathsf{RTEnd}]\quad \frac{\Delta \text{ end-only}}{\epsilon \in \mathcal{R}_{\vec{y}}(\Delta, \vec{y} : [])} \qquad [\mathsf{RTSeq}]\quad \frac{r_1 \in \mathcal{R}_{\vec{y}}(\Delta_1) \qquad r_2 \in \mathcal{R}_{\vec{y}}(\Delta_2)}{r_1 r_2 \in \mathcal{R}_{\vec{y}}(\Delta_1; \Delta_2)}$$

Figure 14: Runs of specifications

- $r = \langle \mathsf{q}, \mathsf{y}\mathsf{d}\rangle r'$, *or else and* $r' \in \mathcal{R}_{\vec{y}}(\Delta')$.

Theorem 8.7 below ensures that well-formed global types are covered by their projections (Definition 6.6 on page 21). Theorem 8.8 ensures that a local specification can be covered by a set of implementations where a role p_i that is played by the same well-typed process P, as is WSI of the role p played by P in the global type. As a corollary we have that a well-typed process P is a WSI of the role it plays.

Theorem 8.7 (Coverage & projections)**.** *Let* $\mathcal{G}(\vec{y}) \triangleq \mathsf{G}$ *be a global type with* $\mathcal{P}(\mathsf{G}) = \{\mathsf{p}_0, \ldots, \mathsf{p}_n\}$ *and* $\Delta = (\vec{y},\mathsf{p}_0) : \mathbb{T}_0, \ldots, (\vec{y},\mathsf{p}_n) : \mathbb{T}_n, \vec{y} : []$ *such that* $\supset(\mathbb{T}_i) = \mathsf{nf}(\mathsf{G}\upharpoonright \mathsf{p}_i)$ *for all* $0 \leq i \leq n$*. Then* $\mathcal{R}(\mathsf{G}) \Subset \mathcal{R}_{\vec{y}}(\Delta)$.

Theorem 8.8 (Typeability & coverage)**.** *Let* $\mathcal{G}(\vec{y}) \triangleq \mathsf{G}$ *be a global type with* $\mathcal{P}(\mathsf{G}) = \{\mathsf{p}_0, \ldots, \mathsf{p}_n\}$ *and* $\Delta = (\vec{y},\mathsf{p}_0) : \mathbb{T}_0, \ldots, (\vec{y},\mathsf{p}_n) : \mathbb{T}_n, \vec{y} : []$ *such that* $\supset(\mathbb{T}_i) = \mathsf{nf}(\mathsf{G}\upharpoonright \mathsf{p}_i)$ *for all* $0 \leq i \leq n$*. If* $e \lrcorner \Gamma \vdash P \triangleright \Delta', (\vec{y},\mathsf{p}_i) : \mathbb{T}_i$ *then the set* $\mathbb{I} = \{\iota_{\vec{y}@u} \mid e \lrcorner \Gamma \vdash \iota_{\vec{y}@u} \triangleright \Delta, \Delta'' \wedge \iota(\mathsf{p}_i) = P\}$ *covers* $\mathcal{R}_{\vec{y}}(\Delta)$*, i.e.,* $\mathcal{R}_{\vec{y}}(\Delta) \Subset \mathcal{R}\mathbb{I}$.

The result above relies on an auxiliary result that shows that each run r of an specification Δ can be covered by a well-typed implementation $\iota_{\vec{y}@u}$ of $\mathcal{G}(\vec{y})$ in which P plays role p (details are provided in Appendix B.4, Lemma B.14). Since $\Subset$ is transitive, we conclude that any well-typed process is a WSI of a role in a choreography.

Corollary 8.9 (WSI of well-typed processes)**.** *Let* $\mathcal{G}(\vec{y})$ *be a global type,* $\mathsf{p} \in \mathcal{P}(\mathcal{G})$*. If* $e \lrcorner \Gamma \vdash P \triangleright \Delta$ *and* $(\vec{y},\mathsf{p}) \in \mathrm{dom}(\Delta)$*,* P *is a whole-spectrum implementation of* p.

9. Conclusion and Related Work

WSI rejects implementations of a role that persistently avoids the execution of some branches in a choreography. Although WSI is defined as a relation between the traces of a global type

and those of its candidate implementations, it can be checked by using multiparty session types. As standard, the soundness of our type system –guaranteed by the conformance of the typing (Theorem 8.4)– ensures that the behaviour of well-typed implementations follows the protocol described by the global type (i.e., global types are interpreted as constraints). Moreover, we show that (*i*) the sets of the projections of a global type $\mathcal{G}$ preserves all the traces in $\mathcal{G}$ (Theorem 8.7); and (*ii*) a well-typed process can be used to obtain any trace of the projections of a global type when interacting in a proper context (Theorem 8.8). These two results and the fact that the covering relation is transitive allow us to conclude that any well-typed process is a WSI of a role in a choreography (Corollary 8.9), i.e., global types are interpreted as obligations.

9.1. **Behavioural types.** In this paper we have followed the rich line of research fostering the application of behavioural types to concurrent programs. Examples of similar approaches are those to guarantee properties of complex concurrent systems such as in the seminal work of Kobayashi on deadlock freedom for the π-calculus [29, 32] or progress analysis for choreographies [16], information flow analysis [31, 8], design-by-contract for message-passing systems [3], or self-adaptation [15, 12]. Our type system is more restrictive than [26, 1, 5, 11, 7] as it rules out sound but not exhaustive implementations, which previous type systems considered well-typed. To the best of our knowledge, the only proposal dealing with complete (i.e., exhaustive) realisations in a behavioural context is [10] but this approach focuses on non-deterministic implementation languages. WSI coincides with projection realisability [33, 42, 10] when implementation languages feature non-deterministic internal choices. On the contrary, WSI provides a finer criterion to distinguish deterministic implementations, as illustrated by the motivating example in the introduction.

Appendix C illustrates that WSI can be recast into other computational models or settings, e.g., in the context of guarded automata. In such context, WSI admits a more succinct characterisation, and hence, would appear as a more amenable definition for WSI (when compared against the definition given in Section 6). However, we remark that several technicalities in our proposal arise when developing a static and modular technique for ensuring WSI. MSTs are perhaps the most widely accepted technique for developing static and modular nal verification techniques for multiparty interactions, and to link their formal specification to programming languages, even at the expenses of dense technical definition (e.g., three different languages, static and run-time semantics, well-formed conditions, etc). It may be the case that similar techniques could be developed for ensuring WSI implementations in other contexts, e.g., guarded automata, but the starting point is not that obvious. It could be the case that some model-checking solution could be developed but the existence of a feasible and simpler approach is an interesting question that requires future work (note that the definitions in Appendix C are based on an existential quantification over possible contexts, which could be problematic for obtaining an effective procedure).

9.2. **WSI and subtyping.** The standard subtyping relation [22] is not suitable for WSI because the liberal elimination of internal choices prevents WSI. For example, standard subtyping for output prefixes allows process $\mathsf{y}(v).\overline{\mathsf{y}_1}\,e$ to have local type $\mathsf{y}\ \mathsf{d}; (\overline{\mathsf{y}_1}\ \mathsf{d} + \overline{\mathsf{y}_2}\ \mathsf{d})$, and this clearly violates WSI. The problematic aspect is that subtyping allows for a liberal implementation of internal choices, whereas WSI requires a precise implementation of all branches in a choice. This means that a WSI of a type T, may not be a WSI of a subtype of

T. The investigation of suitable forms of subtyping for WSIs is scope for future work. To some extent our proposal is related to the fair subtyping approach in [41], where refinement is studied under the fairness assumption: fair subtyping differs from usual subtyping when considering infinite computations but WSI differs from partial implementation also when considering finite computations.

9.3. **Languages for types and processes.** Our notation for session types combines interaction and branching in one single operation. This choice was made for the sake of conciseness. The combination of interaction and branching in one single operation has been used in several works, albeit in different flavours (e.g., [13], [34], [2] and [19]). In [34] the notation is more flexible than ours, allowing (syntactically) global types of the form

$$\mathsf{p} \twoheadrightarrow \mathsf{q}\colon \mathsf{y}_1\ \mathsf{d}_1; \mathsf{G}_1 + \mathsf{r} \twoheadrightarrow \mathsf{s}\colon \mathsf{y}_2\ \mathsf{d}_2; \mathsf{G}_2 \tag{9.1}$$

which our syntax does not allows (as it requires $\mathsf{p} = \mathsf{r}$). However, the work in [34] rules out types as the one in (9.1) when $\mathsf{p} \neq \mathsf{r}$ by the well-formedness rules (i.e., knowledge of choice). Our notation simplifies well-formedness by having this requirement in the syntax. Another line of work [13, 2] uses a syntax more restrictive than ours: in types of the form (9.1), [13, 2] require $\mathsf{p} = \mathsf{r}$ and $\mathsf{q} = \mathsf{s}$. Such restriction prevents types of the form

$$\mathsf{p} \twoheadrightarrow \mathsf{q}\colon \mathsf{y}_1\ \mathsf{d}_1; \mathsf{r} \twoheadrightarrow \mathsf{s}\colon \mathsf{y}_2\ \mathsf{d}_2; \mathsf{end} + \mathsf{r} \twoheadrightarrow \mathsf{s}\colon \mathsf{y}_2\ \mathsf{d}_2; \mathsf{p} \twoheadrightarrow \mathsf{q}\colon \mathsf{y}_1\ \mathsf{d}_1; \mathsf{end}$$

which are instead well-formed in our theory. The notation introduced in this paper differs from usual syntaxes for session types in two other significant ways, which we highlight below. Firstly, we use iteration instead of recursion. In order to verify WSI, our proof system needs to statically determine that the body of each iteration: (1) happens at least once (if interactions in the body are not executed the implementation is not WSI), and (2) terminates (to ensure that interactions in the continuation of an iteration will be executed). For this reason, we need *finitary interactions* in the language for processes. However, usual interpretation of recursive types (where maximal fixpoints are assumed) is not finitary. Therefore, the use of standard recursive session types would introduce a mismatch between the semantics of programs (where we need finitary iterations) with the usual interpretation of recursive types (where maximal fixpoints are assumed).

Therefore, the static verification of WSI requires a form of recursion more restrictive than the one in previous work on session types [26, 1], where the number of iterations is limited. This restriction is on the lines of [10] that also considers finite traces, and is close to for-loops of programming languages. The extension of our theory with a more general form of iteration is scope for future work.

Because of the chosen notation for interaction-choice, iteration requires some care to ensure that all participants agree on whether another iteration should be executed. We do this by requiring that each iteration has exactly one controller-participant as in [10].

Secondly, we use sequential composition to allow clear delimitation of structured constructs (loops and conditional statements) hence ease static verification and its modularity.

We disregarded delegation in our framework because its addition would greatly increase the complexity of our framework. A type system guaranteeing WSI under delegation in our context should not be problematic to device at the cost of a higher technical complexity.

9.4. **Choreography languages.** In the literature, the term 'choreography' is used in two different ways. The first one, which we follow, considers choreographies as abstract specifications [30] or formal models (e.g., Petri Nets [37] or session types [26, 1]) where choices are non-deterministic, in the sense that branches are not associated to conditions (e.g., as in if-then-else statements). The W3C's Web Services Choreography Description Language (WS-CDL) [30] is an XML-based specification language where choices are enumerated without expressing conditions. In the context of multiparty session types [26, 1], choreographies are modelled as global types which, by projection, produce abstractions of processes (with non-deterministic choices in the sense specified above) that can be used for static and dynamic verification. On this thread, we also mention works that study partial vs complete realisations [37, 10].Remarkably, WSI coincides with projection realisability [33, 42, 10] when the language adopted to implement choreographies features non-deterministic internal choices. On the contrary, WSI provides a finer criterion to distinguish deterministic implementations, as illustrated by the motivating example in the introduction. The second usage is in the context of choreographic programming (e.g., [9], [36], [18]) where a choreography is a built in construct for programs (hence deterministic) used to produce correct-by-design code. In such contexts there is no need to resolve non-deterministic choices since the language is deterministic.

9.5. **POP 2 specification.** We captured (to the best of our understanding) the most salient aspects of the communication of the POP2 protocol according to its official informal specification in the state-machines in pages 16–18 of RFC937 [6]. We have focussed on the interaction structure and left out non-functional aspects such as timeouts that are beyond the scope of this paper. We have adopted a simplification on the interaction structure for the sake of a simpler presentation: POP2 has a branch 'quit' from state 'SIZE' (see [6], page 16) whereas our model $\mathsf{G}_{\texttt{SIZE}}$ in Section 5 does not have a 'quit' option. To quit the protocol from state $\mathsf{G}_{\texttt{SIZE}}$ one needs to go back to state $\mathsf{G}_{\texttt{FOLD}}$ (which offers the option 'quit' as also the corresponding state in the RFC). In our formalism, the end of a loop has to be signalled by exactly one message (in our case message 'fold'). We could have encoded this extra 'quit' option from state 'SIZE' by either:

- extending our choreography language to allow a set of possible messages to signal termination of a loop, which would not add considerable challenge but would increase the technicalities in the presentation;
- encoding the exact POP2 pattern by introducing intermediary states with additional messages which would have made the presentation of the protocol itself less clear.

Our goal was to demonstrate that our typing language can model realistic communications, not to provide an analysis of POP2 on its own sake (which we leave as future work), hence our simplification of the protocol.

Acknowledgement

We thank the anonymous reviewers for reading the paper carefully and providing thoughtful comments.

References

[1] L. Bettini, M. Coppo, L. D'Antoni, M. De Luca, M. Dezani-Ciancaglini, and N. Yoshida. Global progress in dynamically interleaved multiparty sessions. In *CONCUR*, volume 5201 of *LNCS*, pages 418–433. Springer, 2008.

[2] L. Bocchi, R. Demangeon, and N. Yoshida. A multiparty multi-session logic. In C. Palamidessi and M. D. Ryan, editors, *TGC 2012*, volume 8191 of *LNCS*, pages 97–111. Springer, 2012.

[3] L. Bocchi, K. Honda, E. Tuosto, and N. Yoshida. A theory of design-by-contract for distributed multiparty interactions. In *CONCUR*, volume 6269 of *LNCS*, pages 162–176, 2010.

[4] L. Bocchi, H. Melgratti, and E. Tuosto. Resolving non-determinism in choreographies. In Z. Shao, editor, *Programming Languages and Systems*, pages 493–512, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[5] M. Bravetti and G. Zavattaro. A theory of contracts for strong service compliance. *Mathematical Structures in Computer Science*, 19(3):601–638, 2009.

[6] M. Butler, J. Postel, D. Chase, J. Goldberger, and J. Reynoldsa. Post office protocol - version 2. RFC 918, available at `http://tools.ietf.org/html/rfc937`, February 1985.

[7] L. Caires and H. T. Vieira. Conversation types. In *ESOP*, volume 5502 of *LNCS*, pages 285–300. Springer, 2009.

[8] S. Capecchi, I. Castellani, and M. Dezani-Ciancaglini. Information flow safety in multiparty sessions. *Mathematical Structures in Computer Science*, 26(8):1352–1394, 2016.

[9] M. Carbone and F. Montesi. Deadlock-freedom-by-design: multiparty asynchronous global programming. In *POPL '13*, pages 263–274. ACM, 2013.

[10] G. Castagna, M. Dezani-Ciancaglini, and L. Padovani. On global types and multi-party session. *Logical Methods in Computer Science*, 8(1), 2012.

[11] G. Castagna and L. Padovani. Contracts for mobile processes. In *CONCUR 2009*, number 5710 in LNCS, pages 211–228, 2009.

[12] I. Castellani, M. Dezani-Ciancaglini, and J. A. Pérez. Self-adaptation and secure information flow in multiparty communications. *Formal Asp. Comput.*, 28(4):669–696, 2016.

[13] T.-C. Chen, L. Bocchi, P.-M. Deniélou, K. Honda, and N. Yoshida. Asynchronous distributed monitoring for multiparty session enforcement. In R. Bruni and V. Sassone, editors, *TGC*, volume 7173 of *Lecture Notes in Computer Science*, pages 25–45. Springer, 2011.

[14] T.-C. Chen and K. Honda. Specifying stateful asynchronous properties for distributed programs. In *CONCUR*, 2012.

[15] M. Coppo, M. Dezani-Ciancaglini, and B. Venneri. Self-adaptive multiparty sessions. *Service Oriented Computing and Applications*, 9(3-4):249–268, 2015.

[16] M. Coppo, M. Dezani-Ciancaglini, N. Yoshida, and L. Padovani. Global progress for dynamically interleaved multiparty sessions. *Mathematical Structures in Computer Science*, 26(2):238–302, 2016.

[17] D. Crocker. Standard for the format of arpa internet text messages. RFC 822, available at `www.ietf.org/rfc/rfc0822.txt`, February 1982.

[18] M. Dalla Preda, M. Gabbrielli, S. Giallorenzo, I. Lanese, and J. Mauro. Dynamic Choreographies: Theory And Implementation. *Logical Methods in Computer Science*, 13:1 – 57, May 2017.

[19] P. Deniélou and N. Yoshida. Multiparty session types meet communicating automata. In H. Seidl, editor, *Programming Languages and Systems - 21st European Symposium on Programming, ESOP 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings*, volume 7211 of *Lecture Notes in Computer Science*, pages 194–213. Springer, 2012.

[20] M. Dezani-Ciancaglini and U. de'Liguoro. Sessions and session types: An overview. In C. Laneve and J. Su, editors, *WS-FM*, volume 6194 of *Lecture Notes in Computer Science*, pages 1–28. Springer, 2009.

[21] X. Fu, T. Bultan, and J. Su. Realizability of conversation protocols with message contents. *Int. J. Web Service Res.*, 2(4):68–93, 2005.

[22] S. Gay and M. Hole. Types and Subtypes for Client-Server Interactions. In *Proc. of ESOP'99*, volume 1576 of *LNCS*, pages 74–90. Springer-Verlag, 1999.

[23] S. Gay and M. Hole. Subtyping for Session Types in the Pi-Calculus. *Acta Informatica*, 42(2/3):191–225, 2005.

[24] R. Guanciale and E. Tuosto. An abstract semantics of the global view of choreographies. In *Proceedings 9th Interaction and Concurrency Experience, ICE 2016, Heraklion, Greece, 8-9 June 2016.*, pages 67–82, 2016.
[25] R. Guanciale and E. Tuosto. Semantics of global views of choreographies. *Journal of Logic and Algebraic Methods in Programming*, 2017. Revised and extended version of [24]. Accepted for publication. To appear; version with proof available at `http://www.cs.le.ac.uk/people/et52/jlamp-with-proofs.pdf`.
[26] K. Honda, N. Yoshida, and M. Carbone. Multiparty asynchronous session types. In G. C. Necula and P. Wadler, editors, *POPL*, pages 273–284. ACM, 2008.
[27] R. Hu, D. Kouzapas, O. Pernet, N. Yoshida, and K. Honda. Type-safe eventful sessions in Java. In *ECOOP 2010*, volume 6183 of *LNCS*, pages 329–353. Springer-Verlag, 2010.
[28] R. Hu and N. Yoshida. Hybrid session verification through endpoint api generation. In *Fundamental Approaches to Software Engineering*, pages 401–418, Berlin, Heidelberg, 2016. Springer.
[29] A. Igarashi and N. Kobayashi. A generic type system for the pi-calculus. *Theor. Comput. Sci.*, 311(1-3):121–163, 2004.
[30] N. Kavantzas, D. Burdett, G. Ritzinger, T. Fletcher, and Y. Lafon. `http://www.w3.org/TR/2004/WD-ws-cdl-10-20041217`, 2004.
[31] N. Kobayashi. Type-based information flow analysis for the pi-calculus. *Acta Inf.*, 42(4-5):291–347, 2005.
[32] N. Kobayashi. A new type system for deadlock-free processes. In *CONCUR 2006 - Concurrency Theory, 17th International Conference, CONCUR 2006, Bonn, Germany, August 27-30, 2006, Proceedings*, volume 4137 of *Lecture Notes in Computer Science*, pages 233–247. Springer, 2006.
[33] I. Lanese, C. Guidi, F. Montesi, and G. Zavattaro. Bridging the gap between interaction-and process-oriented choreographies. In *SEFM*, 2008.
[34] J. Lange and E. Tuosto. Synthesising choreographies from local session types. In M. Koutny and I. Ulidowski, editors, *CONCUR*, volume 7454 of *LNCS*, pages 225–239, 2012.
[35] J. Lange, E. Tuosto, and N. Yoshida. From Communicating Machines to Graphical Choreographies. In *POPL*, pages 221–232, 2015.
[36] A. Lluch-Lafuente, F. Nielson, and H. R. Nielson. Discretionary information flow control for interaction-oriented specifications. In *Logic, Rewriting, and Concurrency - Essays dedicated to José Meseguer on the Occasion of His 65th Birthday*, volume 9200 of *LNCS*, pages 427–450. Springer, 2015.
[37] N. Lohmann and K. Wolf. Decidability results for choreography realization. In G. Kappel, Z. Maamar, and H. R. M. Nezhad, editors, *ICSOC*, volume 7084 of *Lecture Notes in Computer Science*, pages 92–107. Springer, 2011.
[38] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.
[39] M. Neubauer and P. Thiemann. An Implementation of Session Types. In *Practical Aspects of Declarative Languages (PADL)*, volume 3057 of *LNCS*, pages 56–70. Springer, 2004.
[40] R. Neykova, R. Hu, N. Yoshida, and F. Abdeljallal. A session type provider: compile-time API generation of distributed protocols with refinements in f#. In *CC 2018*, pages 128–138. ACM, 2018.
[41] L. Padovani. Fair subtyping for multi-party session types. In *COORDINATION*, volume 6721 of *LNCS*, pages 127–141, 2011.
[42] G. Salaün and T. Bultan. Realizability of choreographies using process algebra encodings. In *Integrated Formal Methods*, 2009.
[43] J. Su, T. Bultan, X. Fu, and X. Zhao. Towards a theory of web service choreographies. In M. Dumas and R. Heckel, editors, *WS-FM*, volume 4937 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2007.

Appendix A. Auxiliary Properties of Typing

In this section we provide technical details and auxiliary properties of the type system, which are used in the proof of the main results of the paper.

A.1. **Normal form $\mathsf{nf}(\_,\_)$.** Below we state useful results about the normal form of pseudo-types. We start by introducing a well-founded relation on pseudo-types, which will be used for inductive proofs. The relation $<$ on pseudo-types is defined in terms of the following function $\omega : \mathbb{T} \to \mathbb{N}$:

$$\begin{aligned}
\omega(e \prec \mathsf{end}) &= 1\\
\omega(\bigoplus_{i\in I} e_i \prec \overline{\mathsf{y}_i}\ \mathsf{d}_i.\mathbb{T}_i) &= 1 + \max\ \{\omega(\mathbb{T}_i)\}_{i\in I}\\
\omega(\sum_{i\in I} e_i \prec \mathsf{y}_i\ \mathsf{d}_i.\mathbb{T}_i) &= 1 + \max\ \{\omega(\mathbb{T}_i)\}_{i\in I}\\
\omega(\mathbb{T}_1;\mathbb{T}_2) &= 2 * \omega(\mathbb{T}_1) + \omega(\mathbb{T}_2)\\
\omega(\mathbb{T}^\star) &= 1 + \omega(\mathbb{T})
\end{aligned}$$

We say $\mathbb{T}_1 < \mathbb{T}_2$ iff $\omega(\mathbb{T}_1) < \omega(\mathbb{T}_2)$. It is straightforward to check that $\omega(\mathbb{T}) > 0$ for all $\mathbb{T}$. Consequently, $(\mathbb{T}, <)$ is well-founded.

Lemma A.1. *$\mathsf{nf}(\mathbb{T})$ is defined for any $\mathbb{T}$ (i.e., it terminates).*

Proof. The proof follows by showing that the function $f(e,\mathbb{T}) = \omega(\mathbb{T})$ is a variant function for the definition of $\mathsf{nf}(e,\mathbb{T})$ in Figure 8. We proceed by case analysis on the equations in Figure 8 (we illustrate the interesting cases below).

(2) $f(e, \bigoplus_{i\in I} e_i \prec \overline{\mathsf{y}_i}\ \mathsf{d}_i.\mathbb{T}_i) = 1 + \max\ \{\omega(\mathbb{T}_i)\}_{i\in I} > \omega(\mathbb{T}_i) = f(e_i \wedge e, \mathbb{T}_i)$ for all $i \in I \setminus J$.

(3) Analogous to (2).

(5)

$$\begin{aligned}
f(e, (\bigoplus_{i\in I} e_i \prec \overline{\mathsf{y}_i}\ \mathsf{d}_i.\mathbb{T}_i);\mathbb{T}) &= \omega((\bigoplus_{i\in I} e_i \prec \overline{\mathsf{y}_i}\ \mathsf{d}_i.\mathbb{T}_i);\mathbb{T})\\
&= 2 * (1 + \max\ \{\omega(\mathbb{T}_i)\}_{i\in I}) + \omega(\mathbb{T})\\
&= 2 + \max\ \{2 * \omega(\mathbb{T}_i) + \omega(\mathbb{T})\}_{i\in I}\\
&> 1 + \max\ \{2 * \omega(\mathbb{T}_i) + \omega(\mathbb{T})\}_{i\in I}\\
&= \omega((\bigoplus_{i\in I} e_i \prec \overline{\mathsf{y}_i}\ \mathsf{d}_i.\mathbb{T}_i;\mathbb{T}))\\
&= f(e, (\bigoplus_{i\in I} e_i \prec \overline{\mathsf{y}_i}\ \mathsf{d}_i.\mathbb{T}_i;\mathbb{T}))
\end{aligned}$$

(6) Analogous to (5).

(7)

$$\begin{aligned}
f(e, (\mathbb{T}_1;\mathbb{T}_2);\mathbb{T}_3) &= \omega((\mathbb{T}_1;\mathbb{T}_2);\mathbb{T}_3)\\
&= 2 * (2 * \omega(\mathbb{T}_1) + \omega(\mathbb{T}_2)) + \omega(\mathbb{T}_3)\\
&= 4 * \omega(\mathbb{T}_1) + 2 * \omega(\mathbb{T}_2) + \omega(\mathbb{T}_3)\\
&> 2 * \omega(\mathbb{T}_1) + 2 * \omega(\mathbb{T}_2) + \omega(\mathbb{T}_3)\\
&= \omega(\mathbb{T}_1;(\mathbb{T}_2;\mathbb{T}_3))\\
&= f(e, \mathbb{T}_1;(\mathbb{T}_2;\mathbb{T}_3))
\end{aligned}$$

□

Lemma A.2. *For all $e, e', \mathbb{T}$, if $\mathsf{nf}(e,\mathbb{T}) = e'' \prec \mathsf{end}$, then $\mathsf{nf}(e \wedge e', \mathbb{T}) = e'' \wedge e' \prec \mathsf{end}$.*

Proof. By well-founded induction on $(\mathbb{T}, <)$. The proof follows by case analysis on the structure of $\mathbb{T}$. □

Lemma A.3. *For all* $e, e', \mathbb{T}$, $\mathsf{nf}(e, \mathsf{nf}(e', \mathbb{T})) = \mathsf{nf}(e \wedge e', \mathbb{T})$.

Proof. By well-founded induction on $(\mathbb{T}, <)$. The proof follows by case analysis on the structure of $\mathbb{T}$.

- $\mathbb{T} = e'' \prec \mathsf{end}$: Then,

$$\begin{array}{rll} \mathsf{nf}(e, \mathsf{nf}(e', \mathbb{T})) & = \mathsf{nf}(e, \mathsf{nf}(e', e'' \prec \mathsf{end})) & \textit{by def. of } \mathbb{T} \\ & = \mathsf{nf}(e, e' \wedge e'' \prec \mathsf{end}) & \textit{by Figure 8(1)} \\ & = e \wedge e' \wedge e'' \prec \mathsf{end} & \textit{by Figure 8(1)} \\ & = \mathsf{nf}(e \wedge e', e'' \prec \mathsf{end}) & \textit{by Figure 8(1)} \\ & = \mathsf{nf}(e \wedge e', \mathbb{T}) & \textit{by def. of } \mathbb{T} \end{array}$$

- $\mathbb{T} = \bigoplus_{i \in I} e_i \prec \overline{\mathsf{y}_i}\ \mathsf{d}_i.\mathbb{T}_i$: Then,

$$\begin{array}{rll} \mathsf{nf}(e, \mathsf{nf}(e', \mathbb{T})) & = \mathsf{nf}(e, \mathsf{nf}(e', \bigoplus_{i \in I} e_i \prec \overline{\mathsf{y}_i}\ \mathsf{d}_i.\mathbb{T}_i)) & \textit{by def. of } \mathbb{T} \\ & = \mathsf{nf}(e, \bigoplus_{i \in I \setminus J'} e_i \wedge e' \prec \overline{\mathsf{y}_i}\ \mathsf{d}_i.\mathsf{nf}(e_i \wedge e', \mathbb{T}_i)) & \textit{by Figure 8(2)} \\ & \qquad J' = \{i \in I \mid (e \wedge e'_i) \iff \mathtt{false}\} \neq I & \\ & = \bigoplus_{i \in (I \setminus J') \setminus J} e_i \wedge e' \wedge e \prec \overline{\mathsf{y}_i}\ \mathsf{d}_i.\mathsf{nf}(e, \mathsf{nf}(e_i \wedge e', \mathbb{T}_i)) & \textit{by Figure 8(2)} \\ & \qquad J = \{i \in I \setminus J' \mid (e \wedge e'_i) \iff \mathtt{false}\} \neq I \setminus J' & \\ & = \bigoplus_{i \in (I \setminus J') \setminus J} e_i \wedge e' \wedge e \prec \overline{\mathsf{y}_i}\ \mathsf{d}_i.\mathsf{nf}(e_i \wedge e' \wedge e, \mathbb{T}_i) & \textit{by ind. hyp.} \\ & = \mathsf{nf}(e \wedge e', \bigoplus_{i \in I} e_i \prec \overline{\mathsf{y}_i}\ \mathsf{d}_i.\mathbb{T}_i) & \textit{by Figure 8(2)} \\ & = \mathsf{nf}(e \wedge e', \mathbb{T}) & \textit{by def. of } \mathbb{T} \end{array}$$

  The cases in which $I = J'$ or $I \setminus J' = J$ follow immediately because $\mathsf{nf}(e, \mathsf{nf}(e', \mathbb{T})) = \mathtt{false} \prec \mathsf{end} = \mathsf{nf}(e \wedge e', \mathbb{T})$.
- $\mathbb{T} = \sum_{i \in I} e_i \prec \mathsf{y}_i\ \mathsf{d}_i.\mathbb{T}_i$: It follows analogously to the previous one.
- $\mathbb{T} = \mathbb{T}_1; \mathbb{T}_2$: We proceed by case analysis on the structure of $\mathbb{T}_1$. The cases $\mathbb{T}_1 = e'' \prec \mathsf{end}$, $\mathbb{T}_1 = \bigoplus_{i \in I} e_i \prec \overline{\mathsf{y}_i}\ \mathsf{d}_i.\mathbb{T}_i$ and $\mathbb{T}_1 = \sum_{i \in I} e_i \prec \mathsf{y}_i\ \mathsf{d}_i.\mathbb{T}_i$ follow analogously to the previous cases.
  The case for $\mathbb{T}_1 = \mathbb{T}'; \mathbb{T}''$ is as follows:

$$\begin{array}{rll} \mathsf{nf}(e, \mathsf{nf}(e', \mathbb{T})) & = \mathsf{nf}(e, \mathsf{nf}(e', (\mathbb{T}'; \mathbb{T}''); \mathbb{T}_2)) & \textit{by def. of } \mathbb{T} \\ & = \mathsf{nf}(e, \mathsf{nf}(e', \mathbb{T}'; (\mathbb{T}''; \mathbb{T}_2))) & \textit{by Figure 8(7)} \\ & = \mathsf{nf}(e \wedge e', \mathbb{T}'; (\mathbb{T}''; \mathbb{T}_2)) & \textit{by ind. hyp.} \\ & = \mathsf{nf}(e \wedge e', (\mathbb{T}'; \mathbb{T}''); \mathbb{T}_2) & \textit{by Figure 8(7)} \\ & = \mathsf{nf}(e \wedge e', \mathbb{T}) & \textit{by def. of } \mathbb{T} \end{array}$$

  The case $\mathbb{T}_1 = (\mathbb{T}')^\star$, is as follows

$$\mathsf{nf}(e, \mathsf{nf}(e', \mathbb{T})) = \mathsf{nf}(e, \mathsf{nf}(e', (\mathbb{T}')^\star; \mathbb{T}_2))$$

There are two cases, when $\mathsf{nf}(e', \mathbb{T}') = e'' \prec \mathsf{end}$, by Figure 8(8)

$$\mathsf{nf}(e, \mathsf{nf}(e', \mathbb{T})) = e \wedge e'' \prec \mathsf{end}$$

Then, the proof is completed by Lemma A.2. Otherwise ($\mathsf{nf}(e', \mathbb{T}') \neq e'' \prec \mathsf{end}$), we proceed as follows

$$\begin{array}{rll}
\mathsf{nf}(e, \mathsf{nf}(e', \mathbb{T})) & = \mathsf{nf}(e, \mathsf{nf}(e', (\mathbb{T}')^\star; \mathbb{T}_2)) & \textit{by def. of } \mathbb{T} \\
& = \mathsf{nf}(e, \mathsf{nf}(e', (\mathbb{T}')^\star); \mathsf{nf}(e', \mathbb{T}_2)) & \textit{by Figure 8(9)} \\
& = \mathsf{nf}(e, \mathsf{nf}(e', \mathbb{T}')^\star; \mathsf{nf}(e', \mathbb{T}_2)) & \textit{by Figure 8(11)} \\
& = \mathsf{nf}(e, \mathsf{nf}(e', \mathbb{T}')^\star); \mathsf{nf}(e, \mathsf{nf}(e', \mathbb{T}_2)) & \textit{by Figure 8(9)} \\
& = \mathsf{nf}(e, \mathsf{nf}(e', \mathbb{T}'))^\star; \mathsf{nf}(e, \mathsf{nf}(e', \mathbb{T}_2)) & \textit{by Figure 8(11)} \\
& = \mathsf{nf}(e \wedge e', \mathbb{T}')^\star; \mathsf{nf}(e \wedge e', \mathbb{T}_2) & \textit{by ind. hyp.} \\
& = \mathsf{nf}(e \wedge e', (\mathbb{T}')^\star); \mathsf{nf}(e \wedge e', \mathbb{T}_2) & \textit{by Figure 8(11)} \\
& = \mathsf{nf}(e \wedge e', (\mathbb{T}')^\star; \mathbb{T}_2) & \textit{by Figure 8(9)} \\
& = \mathsf{nf}(e \wedge e', \mathbb{T}) & \textit{by def. of } \mathbb{T}
\end{array}$$

- $\mathbb{T} = \mathbb{T}_1^\star$: Then,

$$\mathsf{nf}(e, \mathsf{nf}(e', \mathbb{T})) = \mathsf{nf}(e, \mathsf{nf}(e', \mathbb{T}_1^\star))$$

If $\mathsf{nf}(e', \mathbb{T}_1^\star) = e'' \prec \mathsf{end}$, the proof follows by Lemma A.2, otherwise:

$$\begin{array}{rll}
\mathsf{nf}(e, \mathsf{nf}(e', \mathbb{T})) & = \mathsf{nf}(e, \mathsf{nf}(e', \mathbb{T}_1^\star)) & \textit{by def. of } \mathbb{T} \\
& = \mathsf{nf}(e, \mathsf{nf}(e', \mathbb{T}_1^\star)) & \textit{by Figure 8(9)} \\
& = \mathsf{nf}(e, \mathsf{nf}(e', \mathbb{T}_1)^\star) & \textit{by Figure 8(11)} \\
& = \mathsf{nf}(e, \mathsf{nf}(e', \mathbb{T}_1))^\star & \textit{by Figure 8(13)} \\
& = \mathsf{nf}(e \wedge e', \mathbb{T}_1)^\star & \textit{by ind. hyp.} \\
& = \mathsf{nf}(e \wedge e', \mathbb{T}_1^\star) & \textit{by Figure 8(13)} \\
& = \mathsf{nf}(e \wedge e', \mathbb{T}) & \textit{by def. of } \mathbb{T}
\end{array}$$

□

Lemma A.4. *If* $e \iff \texttt{false}$ *then* $\mathsf{nf}(e, \mathbb{T}) = e' \prec \mathsf{end}$ *and* $e' \iff \texttt{false}$.

Proof. By well-founded induction on $(\mathbb{T}, <)$. The proof follows by case analysis on the structure of $\mathbb{T}$. □

Lemma A.5. *For all* e, $\mathbb{T}, \mathbb{T}'$, $\mathsf{nf}(e, \mathbb{T}; \mathsf{nf}(e, \mathbb{T}')) = \mathsf{nf}(e, \mathbb{T}; \mathbb{T}')$.

Proof. By well-founded induction on $(\mathbb{T}, <)$. The proof follows by case analysis on the structure of $\mathbb{T}$.

- $\mathbb{T} = e' \prec \mathsf{end}$: Then,

$$\begin{array}{rll}
\mathsf{nf}(e, \mathbb{T}; \mathsf{nf}(e, \mathbb{T}')) & = \mathsf{nf}(e, e' \prec \mathsf{end}; \mathsf{nf}(e, \mathbb{T}')) & \textit{by def. of } \mathbb{T} \\
& = \mathsf{nf}(e \wedge e', \mathsf{nf}(e, \mathbb{T}')) & \textit{by Figure 8(1)} \\
& = \mathsf{nf}(e \wedge e', \mathbb{T}') & \textit{by Lemma A.3} \\
& = \mathsf{nf}(e, e' \prec \mathsf{end}; \mathbb{T}') & \textit{by Figure 8(1)} \\
& = \mathsf{nf}(e, \mathbb{T}; \mathbb{T}') & \textit{by def. of } \mathbb{T}
\end{array}$$

- $\mathbb{T} = \bigoplus_{i \in I} e_i \prec \overline{\mathsf{y}_i}\ \mathsf{d}_i.\mathbb{T}_i$: Then,

$$\begin{array}{rll}
\mathsf{nf}(e, \mathbb{T}; \mathsf{nf}(e, \mathbb{T}')) & = \mathsf{nf}(e, (\bigoplus_{i \in I} e_i \prec \overline{\mathsf{y}_i}\ \mathsf{d}_i.\mathbb{T}_i); \mathsf{nf}(e, \mathbb{T}')) & \textit{by def. of } \mathbb{T} \\
& = \mathsf{nf}(e, \bigoplus_{i \in I} e_i \prec \overline{\mathsf{y}_i}\ \mathsf{d}_i.\mathbb{T}_i; \mathsf{nf}(e, \mathbb{T}')) & \textit{by Figure 8(6)}
\end{array}$$

The case in which $J = \{i \in I \mid (e \wedge e'_i) \iff \texttt{false}\} = I$ follows immediately. Otherwise, we proceed as follows

$$
\begin{aligned}
&= \bigoplus_{i \in I \setminus J} e_i \wedge e \prec \overline{\mathsf{y}_i}\ \mathsf{d}_i.\mathsf{nf}(e_i \wedge e, \mathbb{T}_i; \mathsf{nf}(e, \mathbb{T}')) && \textit{by Figure 8(2)}\\
&= \bigoplus_{i \in I \setminus J} e_i \wedge e \prec \overline{\mathsf{y}_i}\ \mathsf{d}_i.\mathsf{nf}(e_i, \mathsf{nf}(e, \mathbb{T}_i; \mathsf{nf}(e, \mathbb{T}'))) && \textit{by Lemma A.3}\\
&= \bigoplus_{i \in I \setminus J} e_i \wedge e \prec \overline{\mathsf{y}_i}\ \mathsf{d}_i.\mathsf{nf}(e_i, \mathsf{nf}(e, \mathbb{T}_i; \mathbb{T}')) && \textit{by ind. hyp.}\\
&= \bigoplus_{i \in I \setminus J} e_i \wedge e \prec \overline{\mathsf{y}_i}\ \mathsf{d}_i.\mathsf{nf}(e_i \wedge e, \mathbb{T}_i; \mathbb{T}') && \textit{by Lemma A.3}\\
&= \mathsf{nf}(e, \bigoplus_{i \in I} e_i \prec \overline{\mathsf{y}_i}\ \mathsf{d}_i.\mathbb{T}_i; \mathbb{T}') && \textit{by Figure 8(2)}\\
&= \mathsf{nf}(e, (\bigoplus_{i \in I} e_i \prec \overline{\mathsf{y}_i}\ \mathsf{d}_i.\mathbb{T}_i); \mathbb{T}') && \textit{by Figure 8(6)}\\
&= \mathsf{nf}(e, \mathbb{T}; \mathbb{T}') && \textit{by def. of } \mathbb{T}
\end{aligned}
$$

- $\mathbb{T} = \sum_{i \in I} e_i \prec \mathsf{y}_i\ \mathsf{d}_i.\mathbb{T}_i$: It follows analogously to the previous one.
- $\mathbb{T} = \mathbb{T}_1; \mathbb{T}_2$:

$$
\begin{aligned}
\mathsf{nf}(e, \mathbb{T}; \mathsf{nf}(e, \mathbb{T}')) &= \mathsf{nf}(e, (\mathbb{T}_1; \mathbb{T}_2); \mathsf{nf}(e, \mathbb{T}')) && \textit{by def. of } \mathbb{T}\\
&= \mathsf{nf}(e, \mathbb{T}_1; (\mathbb{T}_2; \mathsf{nf}(e, \mathbb{T}'))) && \textit{by Figure 8(7)}\\
&= \mathsf{nf}(e, \mathbb{T}_1; \mathsf{nf}(e, \mathbb{T}_2; \mathsf{nf}(e, \mathbb{T}'))) && \textit{by ind. hyp.}\\
&= \mathsf{nf}(e, \mathbb{T}_1; \mathsf{nf}(e, \mathbb{T}_2; \mathbb{T}')) && \textit{by ind. hyp.}\\
&= \mathsf{nf}(e, \mathbb{T}_1; (\mathbb{T}_2; \mathbb{T}')) && \textit{by ind. hyp.}\\
&= \mathsf{nf}(e, (\mathbb{T}_1; \mathbb{T}_2); \mathbb{T}') && \textit{by Figure 8(7)}\\
&= \mathsf{nf}(e, \mathbb{T}; \mathbb{T}') && \textit{by def. of } \mathbb{T}
\end{aligned}
$$

- $\mathbb{T} = \mathbb{T}_1^\star$:

$$\mathsf{nf}(e, \mathbb{T}; \mathsf{nf}(e, \mathbb{T}')) = \mathsf{nf}(e, \mathbb{T}_1^\star; \mathsf{nf}(e, \mathbb{T}')) \quad \textit{by def. of } \mathbb{T}$$

If $\mathsf{nf}(e, \mathbb{T}) = e' \prec \mathsf{end}$, then the proof follows straightforwardly from Figure 8(8). Otherwise,

$$
\begin{aligned}
&= \mathsf{nf}(e, \mathbb{T}_1^\star); \mathsf{nf}(e, \mathsf{nf}(e, \mathbb{T}')) && \textit{by Figure 8(9)}\\
&= \mathsf{nf}(e, \mathbb{T}_1^\star); \mathsf{nf}(e, \mathbb{T}') && \textit{by Lemma A.3}\\
&= \mathsf{nf}(e, \mathbb{T}_1^\star; \mathbb{T}') && \textit{by Figure 8(9)}\\
&= \mathsf{nf}(e, \mathbb{T}; \mathbb{T}') && \textit{by def. of } \mathbb{T}
\end{aligned}
$$

□

Lemma A.6. *If* $e \implies e'$, *then* $\mathsf{nf}(e, \mathbb{T}; e' \prec \mathsf{end}) = \mathsf{nf}(e, \mathbb{T})$.

Proof. By well-founded induction on $(\mathbb{T}, <)$. The proof follows by case analysis on the structure of $\mathbb{T}$.

- $\mathbb{T} = e'' \prec \mathsf{end}$: Then,

$$\begin{array}{rll} \mathsf{nf}(e, \mathbb{T}; e' \prec \mathsf{end}) & = \mathsf{nf}(e, e'' \prec \mathsf{end}; e' \prec \mathsf{end}) & \textit{by def. of } \mathbb{T} \\ & = \mathsf{nf}(e \wedge e'', e' \prec \mathsf{end}) & \textit{by Figure 8(6)} \\ & = e \wedge e'' \wedge e' \prec \mathsf{end} & \textit{by Figure 8(1)} \\ & = e \wedge e'' \prec \mathsf{end} & e \implies e' \\ & = \mathsf{nf}(e, e'' \prec \mathsf{end}) & \textit{by Figure 8(6)} \\ & = \mathsf{nf}(e, \mathbb{T}) & \textit{by def. of } \mathbb{T} \end{array}$$

- $\mathbb{T} = \bigoplus_{i \in I} e_i \prec \overline{\mathsf{y}_i}\ \mathsf{d}_i.\mathbb{T}_i$: Then,

$$\begin{array}{rll} \mathsf{nf}(e, \mathbb{T}; e' \prec \mathsf{end}) & = \mathsf{nf}(e, (\bigoplus_{i \in I} e_i \prec \overline{\mathsf{y}_i}\ \mathsf{d}_i.\mathbb{T}_i); e' \prec \mathsf{end}) & \textit{by def. of } \mathbb{T} \\ & = \mathsf{nf}(e, \bigoplus_{i \in I} e_i \prec \overline{\mathsf{y}_i}\ \mathsf{d}_i.(\mathbb{T}_i; e' \prec \mathsf{end})) & \textit{by Figure 8(7)} \end{array}$$

  If $I = J$, then the case follows immediately. Otherwise,

$$\begin{array}{rl} = \bigoplus_{i \in I \setminus J} e_i \wedge e \prec \overline{\mathsf{y}_i}\ \mathsf{d}_i.\mathsf{nf}(e_i \wedge e, \mathbb{T}_i; e' \prec \mathsf{end}) & \textit{by Figure 8(4)} \\ = \bigoplus_{i \in I \setminus J} e_i \wedge e \prec \overline{\mathsf{y}_i}\ \mathsf{d}_i.\mathsf{nf}(e_i \wedge e, \mathbb{T}_i) & \textit{by ind. hyp.} \\ = \mathsf{nf}(e, \bigoplus_{i \in I} e_i \prec \overline{\mathsf{y}_i}\ \mathsf{d}_i.\mathbb{T}_i) & \textit{by Figure 8(4)} \\ = \mathsf{nf}(e, \mathbb{T}) & \textit{by def. of } \mathbb{T} \end{array}$$

- $\mathbb{T} = \sum_{i \in I} e_i \prec \mathsf{y}_i\ \mathsf{d}_i.\mathbb{T}_i$: It follows analogously to the previous one.
- $\mathbb{T} = \mathbb{T}_1; \mathbb{T}_2$: Then,

$$\begin{array}{rll} \mathsf{nf}(e, \mathbb{T}; e' \prec \mathsf{end}) & = \mathsf{nf}(e, (\mathbb{T}_1; \mathbb{T}_2); e' \prec \mathsf{end}) & \textit{by def. of } \mathbb{T} \\ & = \mathsf{nf}(e, \mathbb{T}_1; (\mathbb{T}_2; e' \prec \mathsf{end})) & \textit{by Figure 8(9)} \\ & = \mathsf{nf}(e, \mathbb{T}_1; \mathsf{nf}(e, \mathbb{T}_2; e' \prec \mathsf{end})) & \textit{by Lemma A.5} \\ & = \mathsf{nf}(e, \mathbb{T}_1; \mathsf{nf}(e, \mathbb{T}_2)) & \textit{by ind. hyp.} \\ & = \mathsf{nf}(e, \mathbb{T}_1; \mathbb{T}_2) & \textit{by Figure 8(9)} \\ & = \mathsf{nf}(e, \mathbb{T}) & \textit{by def. of } \mathbb{T} \end{array}$$

- $\mathbb{T} = \mathbb{T}_1^\star$: Then,

$$\begin{array}{rll} \mathsf{nf}(e, \mathbb{T}; e' \prec \mathsf{end}) & = \mathsf{nf}(e, \mathbb{T}_1^\star; e' \prec \mathsf{end}) & \textit{by def. of } \mathbb{T} \\ & = \mathsf{nf}(e, \mathbb{T}_1^\star); \mathsf{nf}(e, e' \prec \mathsf{end}) & \textit{by Figure 8(11)} \\ & = \mathsf{nf}(e, \mathbb{T}_1^\star); \mathsf{nf}(e \wedge e', \mathsf{end}) & \textit{by Figure 8(6)} \\ & = \mathsf{nf}(e, \mathbb{T}_1^\star); \mathsf{nf}(e, \mathsf{end}) & e \implies e' \\ & = \mathsf{nf}(e, \mathbb{T}_1^\star; \mathsf{end}) & \textit{by Figure 8(11)} \\ & = \mathsf{nf}(e, \mathbb{T}; \mathsf{end}) & \textit{by def. of } \mathbb{T} \\ & = \mathsf{nf}(e, \mathbb{T}) & \end{array}$$

□

Lemma A.7. *Equality* $\mathsf{nf}(e, \mathbb{T}; (e' \prec \mathsf{end}; \mathbb{T}')) = \mathsf{nf}(e, \mathbb{T}; \mathbb{T}')$ *holds for all* e *and* e' *such that* $\neg(e \iff \mathtt{false})$ *and* $e \implies e'$.

Proof. By well-founded induction on $(\mathbb{T}, <)$. The proof follows by case analysis on the structure of $\mathbb{T}$.

- $\mathbb{T} = e'' \prec \mathsf{end}$: Then,

$$\begin{array}{lll}
\mathsf{nf}(e, \mathbb{T}; (e' \prec \mathsf{end}; \mathbb{T}')) & = \mathsf{nf}(e, e'' \prec \mathsf{end}; (e' \prec \mathsf{end}; \mathbb{T}')) & \textit{by def. of } \mathbb{T} \\
& = \mathsf{nf}(e \wedge e'', e' \prec \mathsf{end}; \mathbb{T}') & \textit{by Figure 8(6)} \\
& = \mathsf{nf}(e \wedge e'' \wedge e', \mathbb{T}'); & \textit{by Figure 8(6)} \\
& = \mathsf{nf}(e \wedge e'', \mathbb{T}'); & e \wedge e'' \wedge e' \iff e \wedge e'' \\
& = \mathsf{nf}(e, e'' \prec \mathsf{end}; \mathbb{T}') & \textit{by Figure 8(6)} \\
& = \mathsf{nf}(e, \mathbb{T}; \mathbb{T}') & \textit{by def. of } \mathbb{T}
\end{array}$$

- $\mathbb{T} = \bigoplus_{i \in I} e_i \prec \overline{\mathsf{y}_i}\ \mathsf{d}_i.\mathbb{T}_i$: Then,

$$\begin{array}{lll}
\mathsf{nf}(e, \mathbb{T}; (e' \prec \mathsf{end}; \mathbb{T}')) & = \mathsf{nf}(e, (\bigoplus_{i \in I} e_i \prec \overline{\mathsf{y}_i}\ \mathsf{d}_i.\mathbb{T}_i); (e' \prec \mathsf{end}; \mathbb{T}')) & \textit{by def. of } \mathbb{T} \\
& = \mathsf{nf}(e, \bigoplus_{i \in I} e_i \prec \overline{\mathsf{y}_i}\ \mathsf{d}_i.(\mathbb{T}_i; (e' \prec \mathsf{end}; \mathbb{T}'))) & \textit{by Figure 8(7)}
\end{array}$$

  If $I = J$, then the case follows immediately. Otherwise,

$$\begin{array}{lll}
= \bigoplus_{i \in I \setminus J} e_i \wedge e \prec \overline{\mathsf{y}_i}\ \mathsf{d}_i.\mathsf{nf}(e_i \wedge e, (\mathbb{T}_i; (e' \prec \mathsf{end}; \mathbb{T}'))) & & \textit{by Figure 8(4)} \\
= \bigoplus_{i \in I \setminus J} e_i \wedge e \prec \overline{\mathsf{y}_i}\ \mathsf{d}_i.\mathsf{nf}(e_i \wedge e, \mathbb{T}_i; \mathbb{T}') & & \textit{by ind. hyp.} \\
= \mathsf{nf}(e, \bigoplus_{i \in I} e_i \prec \overline{\mathsf{y}_i}\ \mathsf{d}_i.(\mathbb{T}_i; \mathbb{T}')) & & \textit{by Figure 8(4)} \\
= \mathsf{nf}(e, (\bigoplus_{i \in I} e_i \prec \overline{\mathsf{y}_i}\ \mathsf{d}_i.\mathbb{T}_i); \mathbb{T}') & & \textit{by Figure 8(7)} \\
= \mathsf{nf}(e, \mathbb{T}; \mathbb{T}') & & \textit{by def. of } \mathbb{T}
\end{array}$$

- $\mathbb{T} = \sum_{i \in I} e_i \prec \mathsf{y}_i\ \mathsf{d}_i.\mathbb{T}_i$: It follows analogously to the previous one.
- $\mathbb{T} = \mathbb{T}_1; \mathbb{T}_2$: Then,

$$\begin{array}{lll}
\mathsf{nf}(e, \mathbb{T}; (e' \prec \mathsf{end}; \mathbb{T}')) & = \mathsf{nf}(e, (\mathbb{T}_1; \mathbb{T}_2); (e' \prec \mathsf{end}; \mathbb{T}')) & \textit{by def. of } \mathbb{T} \\
& = \mathsf{nf}(e, \mathbb{T}_1; (\mathbb{T}_2; (e' \prec \mathsf{end}; \mathbb{T}'))) & \textit{by Figure 8(9)} \\
& = \mathsf{nf}(e, \mathbb{T}_1; \mathsf{nf}(e, \mathbb{T}_2; (e' \prec \mathsf{end}; \mathbb{T}'))) & \textit{by Lemma A.5} \\
& = \mathsf{nf}(e, \mathbb{T}_1; \mathsf{nf}(e, \mathbb{T}_2; \mathbb{T}')) & \textit{by ind. hyp.} \\
& = \mathsf{nf}(e, \mathbb{T}_1; (\mathbb{T}_2; \mathbb{T}')) & \textit{by Lemma A.5} \\
& = \mathsf{nf}(e, (\mathbb{T}_1; \mathbb{T}_2); \mathbb{T}') & \textit{by Figure 8(9)} \\
& = \mathsf{nf}(e, \mathbb{T}; \mathbb{T}') & \textit{by def. of } \mathbb{T}
\end{array}$$

- $\mathbb{T} = \mathbb{T}_1^\star$: Then,

$$\begin{array}{lll}
\mathsf{nf}(e, \mathbb{T}; (e' \prec \mathsf{end}; \mathbb{T}')) & = \mathsf{nf}(e, \mathbb{T}_1^\star; (e' \prec \mathsf{end}; \mathbb{T}')) & \textit{by def. of } \mathbb{T} \\
& = \mathsf{nf}(e, \mathbb{T}_1^\star); \mathsf{nf}(e, e' \prec \mathsf{end}; \mathbb{T}') & \textit{by Figure 8(11)} \\
& = \mathsf{nf}(e, \mathbb{T}_1^\star); \mathsf{nf}(e \wedge e', \mathbb{T}') & \textit{by Figure 8(6)} \\
& = \mathsf{nf}(e, \mathbb{T}_1^\star); \mathsf{nf}(e, \mathbb{T}') & e \implies e' \\
& = \mathsf{nf}(e, \mathbb{T}_1^\star; \mathbb{T}') & \textit{by Figure 8(11)} \\
& = \mathsf{nf}(e, \mathbb{T}; \mathbb{T}') & \textit{by def. of } \mathbb{T}
\end{array}$$

□

Lemma A.8. $\mathsf{nf}(e, \mathbb{T} \bowtie \mathbb{T}') = \mathsf{nf}(e, \mathbb{T}) \bowtie \mathsf{nf}(e, \mathbb{T}')$.

Proof. By straightforward induction on the derivation of $\mathbb{T} \bowtie \mathbb{T}'$. □

A.2. **Typing.** We write $\mathrm{var}(\mathbb{T})$ to denote the variables appearing in the expressions occurring on the pseudo-type $\mathbb{T}$. It is straightforwardly extended to specifications $\mathrm{var}(\Delta)$.

Lemma A.9. *If* $e \sqcup \Gamma \vdash S \triangleright \Delta$, *then* $\mathrm{var}(\Delta) \subseteq \mathrm{var}(e) \cup \mathtt{fx}(S) \cup \mathtt{bx}(S)$.

Proof. By straightforward structural induction on the typing judgment. □

Lemma A.10. *If* $e \sqcup \Gamma \vdash P \triangleright \Delta$ *then* $e \sqcup \Gamma, x : \mathsf{d} \vdash P \triangleright \Delta$.

Proof. By induction on the structure of the proof $e \sqcup \Gamma \vdash P \triangleright \Delta$. □

Lemma A.11. *If* $e \sqcup \Gamma \vdash P \triangleright \Delta$ *then*

- $\mathtt{fx}(P) \cup \mathtt{fy}(P) \cup \mathtt{fx}(e) \subseteq \Gamma$
- $\forall \mathsf{y} \in \mathtt{fy}(P) : \mathsf{y} \in \mathrm{dom}(\Delta)$.

Proof. By induction of the derivation of $e \sqcup \Gamma \vdash P \triangleright \Delta$ and inspecting the rules in Figure 10 and 11. □

Lemma A.12. *If* $e \sqcup \Gamma \vdash P \triangleright \Delta$ *and* $P \equiv Q$, *then* $e \sqcup \Gamma \vdash Q \triangleright \Delta'$ *and* $\mathsf{nf}(\Delta) = \mathsf{nf}(\Delta')$.

Proof. By induction on the proof of $P \equiv Q$.

- $P; (Q; R) \equiv (P; Q); R$ follows by inductive hypothesis and associativity of ; over pseudo types (Figure 8 (9)).
- $\mathsf{0}; P \equiv P; \mathsf{0} \equiv P$ follows by using Lemma A.7 and Lemma A.6. □

Lemma A.13. *If* $e \sqcup \Gamma \vdash P \triangleright \Delta, (\vec{\mathsf{y}},\mathsf{p}) : \mathbb{T}$ *and* $e \iff \mathtt{false}$, *then* $\mathsf{nf}(\mathbb{T}) = \mathtt{false} \prec \mathsf{end}$.

Proof. Follows by induction on the structure of P. The interesting cases are

- [VSeq]: Then, $P = P_1; P_2$, $e \sqcup \Gamma \vdash P_1 \triangleright \Delta_1$, and $e \sqcup \Gamma \vdash P_2 \triangleright \Delta_2$ and $\Delta_1; \Delta_2$ defined. By inductive hypothesis, if $(\vec{\mathsf{y}},\mathsf{p}) \in \mathrm{dom}(\Delta_i)$ then $\mathsf{nf}(\Delta_i(\vec{\mathsf{y}},\mathsf{p})) = \mathtt{false} \prec \mathsf{end}$ for $i = 1, 2$. Then, the result follows by the definition of ; and rule (4) in Figure 8.
- [VIf]: Then, $P = \mathsf{if}\ e'\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2$, $e \wedge e' \sqcup \Gamma \vdash P_1 \triangleright \Delta_1$, and $e \wedge \neg e' \sqcup \Gamma \vdash P_2 \triangleright \Delta_2$ and $\Delta_1 \bowtie \Delta_2$. By inductive hypothesis, if $(\vec{\mathsf{y}},\mathsf{p}) \in \mathrm{dom}(\Delta_i)$ then $\mathsf{nf}(\Delta_i(\vec{\mathsf{y}},\mathsf{p})) = \mathtt{false} \prec \mathsf{end}$ for $i = 1, 2$. Then, the results follows by definition of $\bowtie$. □

Lemma A.14. *If* $e \sqcup \Gamma \vdash P \triangleright \Delta$, *then for all* e' *there is* Δ' *such that:*

- $\mathrm{dom}(\Delta) = \mathrm{dom}(\Delta')$, *and*
- $e \wedge e' \sqcup \Gamma \vdash P \triangleright \Delta'$, *and*
- *for all* $(\vec{\mathsf{y}},\mathsf{p}) \in \mathrm{dom}(\Delta)$, $\Delta'(\vec{\mathsf{y}},\mathsf{p}) = e \wedge e' \prec \mathsf{end}; \Delta(\vec{\mathsf{y}},\mathsf{p})$.

Proof. Let $\Delta(\vec{\mathsf{y}},\mathsf{p}) = \mathbb{T}$ and $\Delta'(\vec{\mathsf{y}},\mathsf{p}) = \mathbb{T}'$. We show that $\mathsf{nf}(\mathbb{T}') = \mathsf{nf}(e \wedge e' \prec \mathsf{end}; \mathbb{T})$ by induction on the structure of the proof for the judgment $e \sqcup \Gamma \vdash P \triangleright \Delta, (\vec{\mathsf{y}},\mathsf{p}) : \mathbb{T}$. We first assume that $\neg(e \wedge e' \iff \mathtt{false})$, and show

- [VReq], [VAcc]: Follow by inductive hypothesis.

- [VRcv]: Then, $\mathbb{T} = \sum_{i\in I} e \prec \mathsf{y}_i\ \mathsf{d}_i.\mathbb{T}_i$, and $\forall i \in I : e \lrcorner \Gamma, x_i : \mathsf{d}_i \vdash P_i \triangleright \Delta, (\vec{\mathsf{y}},\mathsf{p}) : \mathbb{T}_i$.
  Similarly, $\mathbb{T}' = \sum_{i\in I} e \wedge e' \prec \mathsf{y}_i\ \mathsf{d}_i.\mathbb{T}'_i$ and $\forall i \in I : e \wedge e' \lrcorner \Gamma, x_i : \mathsf{d}_i \vdash P_i \triangleright \Delta', (\vec{\mathsf{y}},\mathsf{p}) : \mathbb{T}'_i$

$$\begin{aligned}
\mathsf{nf}(\mathtt{true}, \mathbb{T}') &= \sum_{i\in I} e \wedge e' \prec \mathsf{y}_i\ \mathsf{d}_i.\mathbb{T}'_i && \textit{by def. of } \mathbb{T}'\\
&= \sum_{i\in I} e \wedge e' \prec \mathsf{y}_i\ \mathsf{d}_i.\mathsf{nf}(e \wedge e', \mathbb{T}'_i) && \textit{by Figure 8(5)}\\
&= \sum_{i\in I} e \wedge e' \prec \mathsf{y}_i\ \mathsf{d}_i.\mathsf{nf}(e \wedge e', e \wedge e' \prec \mathsf{end}; \mathbb{T}_i) && \textit{by ind. hyp.}\\
&= \sum_{i\in I} e \wedge e' \prec \mathsf{y}_i\ \mathsf{d}_i.\mathsf{nf}(e \wedge e', \mathbb{T}_i) && \textit{by Figure 8(6)}\\
&= \mathsf{nf}(e \wedge e', \sum_{i\in I} e \prec \mathsf{y}_i\ \mathsf{d}_i.\mathbb{T}_i) && \textit{by Figure 8(5)}\\
&= \mathsf{nf}(e \wedge e', \mathbb{T}) && \textit{by def. of } \mathbb{T}\\
&= \mathsf{nf}(\mathtt{true}, e \wedge e' \prec \mathsf{end}; \mathbb{T}) && \textit{by Figure 8(6)}
\end{aligned}$$

- [VSend]: Then, $\mathbb{T} = e \prec \overline{\mathsf{y}}\ \mathsf{d}; e \prec \mathsf{end}$ and $\mathbb{T}' = e \wedge e' \prec \overline{\mathsf{y}}\ \mathsf{d}; e \wedge e' \prec \mathsf{end}$.

$$\begin{aligned}
\mathsf{nf}(\mathtt{true}, \mathbb{T}') &= \mathsf{nf}(\mathtt{true}, e \wedge e' \prec \overline{\mathsf{y}}\ \mathsf{d}; e \wedge e' \prec \mathsf{end}) && \textit{by def. of } \mathbb{T}'\\
&= \mathsf{nf}(\mathtt{true}, e \wedge e' \prec \overline{\mathsf{y}}\ \mathsf{d}; \mathsf{nf}(e \wedge e', e \prec \mathsf{end})) && \textit{by Figure 8(6)}\\
&= \mathsf{nf}(e \wedge e', e \prec \overline{\mathsf{y}}\ \mathsf{d}; e \prec \mathsf{end}) && \textit{by Figure 8(4)}\\
&= \mathsf{nf}(e \wedge e', \mathbb{T}) && \textit{by def. of } \mathbb{T}\\
&= \mathsf{nf}(\mathtt{true}, e \wedge e' \prec \mathsf{end}; \mathbb{T}) && \textit{by Figure 8(6)}
\end{aligned}$$

- [VEnd]: Then, $\mathbb{T} = e \prec \mathsf{end}$ and $\mathbb{T}' = e \wedge e' \prec \mathsf{end}$. By normalising,

$$\begin{aligned}
\mathsf{nf}(\mathtt{true}, \mathbb{T}') &= \mathsf{nf}(\mathtt{true}, e \wedge e' \prec \mathsf{end}) && \textit{by def. of } \mathbb{T}'\\
&= \mathsf{nf}(e \wedge e', e \prec \mathsf{end}); && \textit{by Figure 8(1)}\\
&= \mathsf{nf}(e \wedge e', \mathbb{T}); && \textit{by def. of } \mathbb{T}'\\
&= \mathsf{nf}(\mathtt{true}, e \wedge e' \prec \mathsf{end}; \mathbb{T}); && \textit{by Figure 8(6)}
\end{aligned}$$

- [VSeq] There are two cases:
  - $(\vec{\mathsf{y}},\mathsf{i}) \in (\mathrm{dom}(\Delta_1) \cup \mathrm{dom}(\Delta_2))$: Then $\mathbb{T} = \mathbb{T}_1; \mathbb{T}_2$ and $\mathbb{T}' = \mathbb{T}'_1; \mathbb{T}'_2$. By inductive hypothesis, $\mathbb{T}'_1 = e \wedge e' \prec \mathsf{end}; \mathbb{T}_1$ and $\mathbb{T}'_2 = e \wedge e' \prec \mathsf{end}; \mathbb{T}_2$.

$$\begin{aligned}
\mathsf{nf}(\mathtt{true}, \mathbb{T}') &= \mathsf{nf}(\mathtt{true}, \mathbb{T}'_1; \mathbb{T}'_2) && \textit{by def. of } \mathbb{T}'\\
&= \mathsf{nf}(\mathtt{true}, (e \wedge e' \prec \mathsf{end}; \mathbb{T}_1); (e \wedge e' \prec \mathsf{end}; \mathbb{T}_2)) && \textit{by ind. hyp.}\\
&= \mathsf{nf}(\mathtt{true}, e \wedge e' \prec \mathsf{end}; (\mathbb{T}_1; (e \wedge e' \prec \mathsf{end}; \mathbb{T}_2))) && \textit{by Figure 8(6)}\\
&= \mathsf{nf}(e \wedge e', \mathbb{T}_1; (e \wedge e' \prec \mathsf{end}; \mathbb{T}_2)) && \textit{by Figure 8(9)}\\
&= \mathsf{nf}(e \wedge e', \mathbb{T}_1; \mathbb{T}_2) && \textit{by Lemma A.7}\\
&= \mathsf{nf}(e \wedge e', \mathbb{T}) && \textit{by def. of } \mathbb{T}\\
&= \mathsf{nf}(\mathtt{true}, e \wedge e' \prec \mathsf{end}; \mathbb{T}) && \textit{by Figure 8(6)}
\end{aligned}$$

  - $(\vec{\mathsf{y}},\mathsf{i}) \notin \mathrm{dom}(\Delta_2)$: Then $\mathbb{T} = \mathbb{T}_1$ and $\mathbb{T}' = \mathbb{T}'_1$. By inductive hypothesis, $\mathbb{T}'_1 = e \wedge e' \prec \mathsf{end}; \mathbb{T}_1$. Hence, $\mathbb{T}' = e \wedge e' \prec \mathsf{end}; \mathbb{T}$.

- [VIf]: Then $\mathbb{T} = \mathbb{T}_1 \bowtie \mathbb{T}_2$ and $\mathbb{T}' = \mathbb{T}'_1 \bowtie \mathbb{T}'_2$.

$$\begin{aligned}
\mathsf{nf}(\mathtt{true}, \mathbb{T}') &= \mathsf{nf}(\mathtt{true}, \mathbb{T}'_1 \bowtie \mathbb{T}'_2) && \textit{by def. of } \mathbb{T}' \\
&= \mathsf{nf}(\mathtt{true}, \mathbb{T}'_1) \bowtie \mathsf{nf}(\mathtt{true}, \mathbb{T}'_2) && \textit{by Lemma A.8} \\
&= \mathsf{nf}(\mathtt{true}, (e \wedge e' \prec \mathsf{end}; \mathbb{T}_1)) \bowtie \mathsf{nf}(\mathtt{true}, (e \wedge e' \prec \mathsf{end}; \mathbb{T}_2)) && \textit{by ind. hyp.} \\
&= \mathsf{nf}(e \wedge e', \mathbb{T}_1) \bowtie \mathsf{nf}(e \wedge e', \mathbb{T}_2) && \textit{by Figure 8(6)} \\
&= \mathsf{nf}(e \wedge e', \mathbb{T}'_1 \bowtie \mathbb{T}'_2) && \textit{by Lemma A.8} \\
&= \mathsf{nf}(e \wedge e', \mathbb{T}) && \textit{by def. of } \mathbb{T} \\
&= \mathsf{nf}(\mathtt{true}, e \wedge e' \prec \mathsf{end}; \mathbb{T}) && \textit{by Figure 8(6)}
\end{aligned}$$

- [VFor]: Then $\mathbb{T} = (\mathbb{T}_1)^\star$, $\mathbb{T}' = (\mathbb{T}'_1)^\star$. Then,

$$\begin{aligned}
\mathsf{nf}(\mathtt{true}, \mathbb{T}') &= \mathsf{nf}(\mathtt{true}, (\mathbb{T}'_1)^\star) && \textit{by def. of } \mathbb{T}' \\
&= \mathsf{nf}(\mathtt{true}, \mathbb{T}'_1)^\star && \textit{by Figure 8(13)} \\
&= \mathsf{nf}(\mathtt{true}, e \wedge e' \prec \mathsf{end}; \mathbb{T}_1)^\star && \textit{by ind. hyp.} \\
&= \mathsf{nf}(e \wedge e', \mathbb{T}_1)^\star && \textit{by Figure 8(6)} \\
&= \mathsf{nf}(e \wedge e', (\mathbb{T}_1)^\star) && \textit{by Figure 8(13)} \\
&= \mathsf{nf}(e \wedge e', \mathbb{T}) && \textit{by def. of } \mathbb{T} \\
&= \mathsf{nf}(\mathtt{true}, e \wedge e' \prec \mathsf{end}; \mathbb{T}) && \textit{by Figure 8(6)}
\end{aligned}$$

- [VForEnd]: It follows analogously to the case [VEnd].
- [Vloop]: Since Δ_1 and Δ_2 are passively compatible, then $\mathrm{dom}(\Delta_1) = \mathrm{dom}(\Delta_2)$, then $(\vec{y}, \mathsf{i}) \in (\mathrm{dom}(\Delta_1) \cap \mathrm{dom}(\Delta_2))$. Consequently, $\mathbb{T} = (\mathbb{T}_1)^\star; \mathbb{T}_2$, $\mathbb{T}' = (\mathbb{T}'_1)^\star; \mathbb{T}_2$. By inductive hypothesis, $\mathbb{T}'_1 = e \wedge e' \prec \mathsf{end}; \mathbb{T}_1$ and $\mathbb{T}'_2 = e \wedge e' \prec \mathsf{end}; \mathbb{T}_2$.

$$\begin{aligned}
\mathsf{nf}(\mathtt{true}, \mathbb{T}') &= \mathsf{nf}(\mathtt{true}, (\mathbb{T}'_1)^\star; \mathbb{T}'_2) && \textit{by def. of } \mathbb{T}' \\
&= \mathsf{nf}(\mathtt{true}, (e \wedge e' \prec \mathsf{end}; \mathbb{T}_1)^\star; (e \wedge e' \prec \mathsf{end}; \mathbb{T}_2)) && \textit{by ind. hyp.} \\
&= \mathsf{nf}(\mathtt{true}, (e \wedge e' \prec \mathsf{end}; \mathbb{T}_1)^\star); \mathsf{nf}(\mathtt{true}, e \wedge e' \prec \mathsf{end}; \mathbb{T}_2) && \textit{by Figure 8(11)} \\
&= \mathsf{nf}(\mathtt{true}, (e \wedge e' \prec \mathsf{end}; \mathbb{T}_1))^\star; \mathsf{nf}(\mathtt{true}, e \wedge e' \prec \mathsf{end}; \mathbb{T}_2) && \textit{by Figure 8(13)} \\
&= \mathsf{nf}(e \wedge e', \mathbb{T}_1)^\star; \mathsf{nf}(e \wedge e', \mathbb{T}_2) && \textit{by Figure 8(6)} \\
&= \mathsf{nf}(e \wedge e', \mathbb{T}_1^\star); \mathsf{nf}(e \wedge e', \mathbb{T}_2) && \textit{by Figure 8(13)} \\
&= \mathsf{nf}(e \wedge e', (\mathbb{T}_1)^\star; \mathbb{T}_2) && \textit{by Figure 8(11)} \\
&= \mathsf{nf}(e \wedge e', \mathbb{T}) && \textit{by def. of } \mathbb{T} \\
&= \mathsf{nf}(\mathtt{true}, e \wedge e' \prec \mathsf{end}; \mathbb{T}) && \textit{by Figure 8(6)}
\end{aligned}$$

If $e \wedge e' \iff \mathtt{false}$, we proceed as follows

$$\begin{aligned}
\mathsf{nf}(\mathtt{true}, e \wedge e' \prec \mathsf{end}; \mathbb{T}) &= \mathsf{nf}(e \wedge e', \mathbb{T}) && \textit{by Figure 8(6)} \\
&= \mathtt{false} \prec \mathsf{end} && \textit{by Lemma A.4} \\
&= \mathbb{T}' && \textit{by Lemma A.13}
\end{aligned}$$

□

Lemma A.15. *If* $e \lrcorner \Gamma \vdash P \triangleright \Delta$, *then*

- $e \wedge e' \lrcorner \Gamma \vdash P \triangleright \Delta'$; *and*
- *for all* $(\vec{y}, \mathsf{p}) \in \mathrm{dom}(\Delta')$, $\Delta'(\vec{y}, \mathsf{p}) = \mathsf{nf}(e', \Delta(\vec{y}, \mathsf{p}))$

Proof. Directly from Lemma A.14. □

A.3. **Consistency.**

Lemma A.16. *If $\sigma \ltimes \mathbb{T}$, then $\sigma' \ltimes \mathbb{T}$ for any σ' such that $\sigma(x) = \sigma'(x)$ for all $x \in \mathrm{var}(\mathbb{T})$.*

Proof. By structural induction on the structure of $\sigma \ltimes \mathbb{T}$ and by noticing that $e \downarrow (\sigma \cap \sigma') = e \downarrow \sigma$ for every expression e in $\mathbb{T}$. □

Lemma A.17. *If $x \notin \mathrm{var}(\mathbb{T})$ and $\sigma \ltimes \mathbb{T}$ then $\sigma|_{\mathrm{dom}(\sigma)\setminus\{x\}} \ltimes \mathbb{T}$.*

Proof. It follows from Lemma A.16. □

Lemma A.18. *If $\mathbb{T}\bowtie\mathbb{T}'$ is defined and either $\sigma \ltimes \mathbb{T}$ or $\sigma \ltimes \mathbb{T}'$, then $\sigma \ltimes (\mathbb{T}\bowtie\mathbb{T}')$.*

Proof. By induction on the structure of $\mathbb{T}$. □

Lemma A.19. *Let σ, Δ, e, and Γ be such that conditions (1)–(3) in Definition 8.1 hold. Then*

$$e \llcorner \Gamma \vdash P \triangleright \Delta \quad \Longrightarrow \quad \forall (\vec{\mathsf{y}},\mathsf{p}) \in \mathrm{dom}(\Delta) : \ \sigma \ltimes \Delta(\vec{\mathsf{y}},\mathsf{p})$$

Proof. By structural induction on the derivation of the typing judgement. We proceed by case analysis on the last rule applied in the derivation of the judgment $e \llcorner \Gamma \vdash P \triangleright \Delta$.

- [VReq] The thesis directly follows from the inductive hypothesis.
- [VAcc] The thesis directly follows from the inductive hypothesis.
- [VRcv] Then, $\Delta = \Delta', (\vec{\mathsf{y}},\mathsf{p}) : \sum_{i\in I} e \prec \mathsf{y}_i\ \mathsf{d}_i.\mathbb{T}_i$. The inductive hypothesis applied to the premiss of [VRcv] implies that $\sigma, x_i \mapsto \mathsf{v}_i \ltimes \mathbb{T}_i$ for all $i \in I$ and that $\sigma, x_i \mapsto \mathsf{v}_i \ltimes \Delta'(\vec{\mathsf{y}'},\mathsf{q})$. Since $e \downarrow \sigma = \mathtt{true}$, then $e \downarrow \sigma, x_i \mapsto \mathsf{v}_i = \mathtt{true}$ for all $i \in I$. By inductive hypothesis on all the premiss, we conclude that $\sigma, x_i \mapsto \mathsf{v}_i \ltimes \mathbb{T}_i$. By Lemma A.17, $\sigma \ltimes \mathbb{T}_i$ for all $i \in I$, so we can apply rule [CRcv] and obtain the thesis.
- [VSend] Then, $\Delta = \Delta', (\vec{\mathsf{y}},\mathsf{p}) : e \prec \overline{\mathsf{y}}\ \mathsf{d}; e \prec \mathsf{end}$ and, by the premiss of [VSend], $\Delta'(\vec{\mathsf{y}'},\mathsf{q}) = e \prec \mathsf{end}$ and $\sigma \ltimes e \prec \mathsf{end}$ by rule [CRcv]. We just need to prove that $\sigma \ltimes e \prec \overline{\mathsf{y}}\ \mathsf{d}$ so to apply rule [CSeq] and obtain the thesis. This is indeed the case as $e \downarrow \sigma = \mathtt{true}$ by assumption (3) of Definition 8.1 which implies $e\&\sigma$ and so, by rule [CSend] we conclude the proof.
- [VEnd] Then, $(\vec{\mathsf{y}},\mathsf{p}) \in \mathrm{dom}(\Delta)$ and $\Delta(\vec{\mathsf{y}},\mathsf{p}) = e \prec \mathsf{end}$. Moreover, $e \downarrow \sigma = \mathtt{true}$ by assumption. Therefore, $\sigma \ltimes e \prec \mathsf{end}$ by rule [CRcv].
- [VSeq] We have

$$\frac{e \llcorner \Gamma \vdash P_1 \triangleright \Delta_1 \qquad e \llcorner \Gamma \vdash P_2 \triangleright \Delta_2}{e \llcorner \Gamma \vdash P_1; P_2 \triangleright \Delta_1; \Delta_2} \quad \text{with } \Delta = \Delta_1; \Delta_2$$

  For all $(\vec{\mathsf{y}},\mathsf{p}) \notin \mathrm{dom}(\Delta_1) \cap \mathrm{dom}(\Delta_2)$ the thesis follows directly by the inductive hypothesis on one of the premiss of the rule above. If $(\vec{\mathsf{y}},\mathsf{p}) \in \mathrm{dom}(\Delta_1) \cap \mathrm{dom}(\Delta_2)$ then $\Delta(\vec{\mathsf{y}},\mathsf{p}) = \Delta_1(\vec{\mathsf{y}},\mathsf{p}); \Delta_2(\vec{\mathsf{y}},\mathsf{p})$ and, by inductive hypothesis, both $\sigma \ltimes \Delta_1(\vec{\mathsf{y}},\mathsf{p})$ and $\sigma \ltimes \Delta_2(\vec{\mathsf{y}},\mathsf{p})$ hold. Hence, the thesis follows by rule [CSeq].
- [VIf] We have

$$\frac{e \wedge e' \llcorner \Gamma \vdash P_1 \triangleright \Delta_1 \qquad e \wedge \neg e' \llcorner \Gamma \vdash P_2 \triangleright \Delta_2}{e \llcorner \Gamma \vdash \mathsf{if}\ e'\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2 \triangleright \Delta_1 \bowtie \Delta_2} \quad \text{with } \Delta = \Delta_1 \bowtie \Delta_2$$

  Since $e \downarrow \sigma = \mathtt{true}$, we have either $(e \wedge e') \downarrow \sigma = \mathtt{true}$ or $(e \wedge \neg e') \downarrow \sigma = \mathtt{true}$; hence, we can apply the inductive hypothesis to one of the premisise of the rule above. So, we have that $\sigma \ltimes \mathbb{T}_1$ or $\sigma \ltimes \mathbb{T}_2$ and $\sigma \ltimes \mathbb{T}_1 \bowtie \mathbb{T}_2$ hold by Lemma A.18.

- [VFor] We have that $\Delta = \Delta_1^*$ for some specification Δ_1 such that $e \wedge x \in \ell \sqcup x : \mathsf{d}, \Gamma \vdash P \triangleright \Delta_1$. By the inductive hypothesis, we have $\sigma, x \mapsto \mathsf{v} \ltimes \Delta_1(\vec{\mathsf{y}},\mathsf{p})$ for all $(\vec{\mathsf{y}},\mathsf{p}) \in \mathrm{dom}(\Delta_1) = \mathrm{dom}(\Delta)$. The thesis then follows by Lemma A.17 and applying rule [CLoop].
- [VForEnd] Similarly to the case [VEnd].
- [Vloop] We have that $\Delta = \Delta_1^*; \Delta_2$ for two specifications Δ_1 and Δ_2 such that

$$\frac{e \sqcup \Gamma \vdash N \triangleright \Delta_1 \qquad e \sqcup \Gamma \vdash M \triangleright \Delta_2 \qquad \Delta_1 \text{ and } \Delta_2 \text{ passively compatible}}{e \sqcup \Gamma \vdash \mathsf{repeat}\ N\ \mathsf{until}\ M \triangleright \Delta_1^\star; \Delta_2}$$

  For all $(\vec{\mathsf{y}},\mathsf{p}) \in \mathrm{dom}(\Delta_1) \cup \mathrm{dom}(\Delta_2) = \mathrm{dom}(\Delta)$, we have $\sigma \ltimes (\vec{\mathsf{y}},\mathsf{p})$ by the inductive hypothesis. The thesis follows by using rules [CLoop] and [CSeq]. □

Lemma A.20. *If $e \sqcup \Gamma \vdash P \triangleright \Delta$, $\sigma \ltimes (e; \Gamma; P; \Delta)$ and Δ active, then $\langle P, \sigma\rangle \xrightarrow{\alpha} \langle P', \sigma'\rangle$ implies $\alpha \neq \mathsf{yd}$.*

Proof. By induction on the structure of the typing judgement.

- [VReq]: Then, $P = \overline{u}^{\mathsf{n}}(\vec{\mathsf{y}}).P'$. By inspecting reduction rules, the only possibility is $\alpha = \overline{u}^{\mathsf{n}}(\vec{\mathsf{y}})$.
- [VAcc]: Then, $P = u^{\mathsf{i}}(\vec{\mathsf{y}}).P'$. By inspecting reduction rules, the only possibility is $\alpha = u^{\mathsf{i}}(\vec{\mathsf{y}})$.
- [VSend]: Then, $P = \overline{\mathsf{y}}\, e'$. By inspecting reduction rules, the only possibility is $\alpha = \overline{\mathsf{y}}\mathsf{v}$.
- [VEnd]: Then, $P = 0$. Hence, there is no reduction.
- [VSeq]: Then, $P = P_1; P_2$, $\Delta = \Delta_1; \Delta_2$, and $e \sqcup \Gamma \vdash P_1 \triangleright \Delta_1$. Note that $\langle P, \sigma\rangle \xrightarrow{\alpha} \langle P', \sigma'\rangle$ implies $\langle P_1, \sigma\rangle \xrightarrow{\alpha} \langle P_1', \sigma'\rangle$. Moreover, Δ active implies Δ_1 active. Then, by inductive hypothesis, $\alpha \neq \overline{\mathsf{y}}\mathsf{d}$.
- [VIf]: Then, $P = \mathsf{if}\ e'\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2$, $\Delta = \Delta_1 \bowtie \Delta_2$. Moreover, $\langle P, \sigma\rangle \xrightarrow{\alpha} \langle P', \sigma'\rangle$ implies $\langle P_1, \sigma\rangle \xrightarrow{\alpha} \langle P_1', \sigma'\rangle$ or $\langle P_1, \sigma\rangle \xrightarrow{\alpha} \langle P_1', \sigma'\rangle$. Moreover, Δ active implies Δ_1 and Δ_2 active. Then, by inductive hypothesis, $\alpha \neq \overline{\mathsf{y}}\mathsf{d}$.
- [VRcv], [VEnd], [VFor], [VForEnd] and [Vloop] implies Δ not active, hence the cases trivially hold. □

Lemma A.21. *Let $e \sqcup \Gamma \vdash P \triangleright \Delta$ and $\vec{\mathsf{y}} \in \mathrm{dom}(\Delta)$. If $\Delta \xrightarrow{\alpha} \Delta'$ with $\alpha \neq \tau$ and $\mathtt{n}(\alpha) \subseteq \vec{\mathsf{y}}$, then for all stores σ such that $\sigma \ltimes (e; \Gamma; P; \Delta)$ the following conditions hold*

(1) $\langle P, \sigma\rangle \xrightarrow{E \vdash \widetilde{\beta}} \xrightarrow{e' \vdash \alpha} \langle P', \sigma'\rangle$ *with* $\mathtt{n}(\widetilde{\beta}) \cap \vec{\mathsf{y}} = \emptyset$
(2) $e \wedge E \wedge e' \sqcup \Gamma' \vdash P' \triangleright \Delta''$ *with* $\Gamma \subseteq \Gamma'$
(3) $\Delta'(\vec{\mathsf{y}},\mathsf{p}) = \Delta''(\vec{\mathsf{y}},\mathsf{p})$ *for all* $(\vec{\mathsf{y}},\mathsf{p}) \in \mathrm{dom}(\Delta)$
(4) $\sigma' \ltimes (\Delta''; e \wedge E \wedge e'; P'; \Gamma')$.

Proof. The proof follows by induction on the structure of the derivation of $e \sqcup \Gamma \vdash P \triangleright \Delta$. We report the representative cases.

- [VReq]Then, $P = \overline{u}^{\mathsf{n}}(\vec{\mathsf{y}}).P''$, $\Delta(u) \equiv \mathcal{G}(\vec{\mathsf{y}})$ and $e \sqcup \Gamma \vdash P'' \triangleright \Delta, (\vec{\mathsf{y}},0) : \mathbb{T}$ with $\mathcal{D}(\mathbb{T}) = \mathcal{D}(\mathcal{G}(\vec{\mathsf{y}})\restriction 0)$. There are two cases:
  (1) $\alpha = \overline{u}^{\mathsf{n}}(\vec{\mathsf{y}})$. Then, $\Delta' = \Delta, (\vec{\mathsf{y}},0) : \mathbb{T}$. Take $P' = P''$, $\widetilde{\beta} = \epsilon$, $E = e' = \mathtt{true}$, $\sigma' = \sigma[\vec{\mathsf{y}} \mapsto u]$, $\Gamma' = \Gamma$ and $\Delta'' = \Delta'$. Note that $\langle P, \sigma\rangle \xrightarrow{e' \vdash \alpha} \langle P', \sigma'\rangle$ by [SReq]. Condition $\sigma' \ltimes (\Delta'; e; P''; \Gamma)$ follows straightforwardly from $\sigma \ltimes (\Delta; e; P; \Gamma)$.
  (2) $\alpha \neq \overline{u}^{\mathsf{n}}(\vec{\mathsf{y}})$. Hence, $\mathtt{n}(\alpha) \cap \vec{\mathsf{y}} = \emptyset$. By Lemma B.1, $\Delta \xrightarrow{\alpha} \Delta'$ implies $\Delta, (\vec{\mathsf{y}},0) : \mathbb{T} \xrightarrow{\alpha} \Delta', (\vec{\mathsf{y}},0) : \mathbb{T}$. By inductive hypothesis on $e \sqcup \Gamma \vdash P'' \triangleright \Delta, (\vec{\mathsf{y}},0) : \mathbb{T}$ and $\Delta, (\vec{\mathsf{y}},0) : \mathbb{T} \xrightarrow{\alpha} \Delta', (\vec{\mathsf{y}},0) : \mathbb{T}$, we conclude that $\langle P'', \sigma[\vec{\mathsf{y}} \mapsto u]\rangle \xrightarrow{E' \vdash \widetilde{\beta}'} \xrightarrow{e' \vdash \alpha} \langle P', \sigma'\rangle$.

Then, take $\widetilde{\beta} = \overline{u}^{\mathsf{n}}(\vec{y})\widetilde{\beta}'$ and $E = E'$. Conditions (2)–(4) follow straightforwardly from inductive hypothesis.

- [TRcv] Then, $P = \sum_{i\in I} \mathsf{y}_i(x_i).P_i$, and $\Delta = (\vec{y},\mathsf{p}) : \sum_{i\in I} e \prec \mathsf{y}_i\ \mathsf{d}_i.\mathbb{T}_i, \Delta_0$ and $\forall i \in I :\ \mathsf{y}_i \in \vec{y}$ and $e \frown \Gamma, x_i : \mathsf{d}_i \vdash P_i \triangleright \Delta, (\vec{y},\mathsf{p}) : \mathbb{T}_i$.
  There are two cases,
  (1) $\alpha = \mathsf{y}_j\mathsf{d}_j$ with $j \in I$. Then, $\Delta' = \Delta, (\vec{y},\mathsf{p}) : \mathbb{T}_j$. The proof is completed by taking $P' = P_j$, and $\widetilde{\beta} = \epsilon$, and $E = e' = \mathtt{true}$, and $\sigma' = \sigma[x_j \mapsto \mathsf{v}]$, and $\Gamma' = \Gamma, x_i : \mathsf{d}_i$, and $\Delta'' = \Delta'$. Note that $\langle P, \sigma\rangle \xrightarrow{e'\vdash\alpha} \langle P', \sigma'\rangle$ by [SRec]. Condition $\sigma' \ltimes (\Delta''; e; P'; \Gamma')$ follows straightforwardly from $\sigma \ltimes (\Delta; e; P; \Gamma)$.
  (2) $\alpha \neq \overline{u}^{\mathsf{n}}(\vec{y})$. Hence, $\mathtt{n}(\alpha) \cap \vec{y} = \emptyset$. The case follows analogously to rule [VReq]. □

Lemma A.22. *If* $\langle P, \sigma\rangle \xrightarrow{e\vdash\alpha} \langle P', \sigma'\rangle$, *then* $e \downarrow \sigma = \mathtt{true}$.

Proof. By straightforward induction on the structure of the proof $\langle P, \sigma\rangle \xrightarrow{e'\vdash\alpha} \langle P', \sigma'\rangle$. □

A.4. **Subject reduction.** We first state an auxiliary property about the semantics of systems.

Lemma A.23. *If* $\langle S, \sigma\rangle \xrightarrow{e\vdash\alpha} \langle S', \sigma'\rangle$ *then* $\mathrm{dom}(\sigma) \subseteq \mathrm{dom}(\sigma')$.

Proof. By induction on the derivation of $\langle S, \sigma\rangle \xrightarrow{e\vdash\alpha} \langle S', \sigma'\rangle$ inspecting the rules in Figure 4 and 5. □

Theorem 8.2 (Subject reduction)**.** *If* $e \frown \Gamma \vdash S \triangleright \Delta$, $\sigma \ltimes (e; \Gamma; S; \Delta)$, *and* $\langle S, \sigma\rangle \xrightarrow{e'\vdash\alpha} \langle S', \sigma'\rangle$ *then there exist* Γ' *and* Δ' *such that*

(1) *if* $\alpha = \mathsf{y}\mathsf{v}$ *then* $\Delta \xrightarrow{\mathsf{y}\mathsf{d}} \Delta'$ *for a sort* d*; moreover, if* $\vdash \mathsf{v} : \mathsf{d}$ *then there is* $x \in \mathbb{X}$ *such that*
$$e \wedge e' \frown \Gamma', x : \mathsf{d} \vdash S' \triangleright \Delta', \qquad \sigma'(x) = \mathsf{v}, \quad \text{and} \quad \sigma' \ltimes (e \wedge e'; \Gamma', x : \mathsf{d}; S'; \Delta')$$
(2) *if* $\alpha = \overline{\mathsf{y}}\mathsf{v}$ *then* $\Delta \xrightarrow{\overline{\mathsf{y}}\mathsf{d}} \Delta'$ *with* $\vdash \mathsf{v} : \mathsf{d}$, $e \wedge e' \frown \Gamma' \vdash S' \triangleright \Delta'$, *and* $\sigma' \ltimes (e \wedge e'; \Gamma'; S'; \Delta')$
(3) *otherwise* $\Delta \xrightarrow{\alpha} \Delta'$, *and* $e \wedge e' \frown \Gamma' \vdash S' \triangleright \Delta'$, $\sigma' \ltimes (e \wedge e'; \Gamma'; S'; \Delta')$.

Proof. The proof is by induction on the derivation of $\langle S, \sigma\rangle \xrightarrow{e'\vdash\alpha} \langle S', \sigma'\rangle$ which may end with the application of one of the rules in Figure 4 (on page 15) or in Figure 5 (on page 15). It is a simple observation that if $\alpha \neq \tau$ then S must be a process, so the proof ends with one of the rules in Figure 4.

Base cases.

- [SReq] In this case $\alpha = \overline{u}^{\mathsf{n}}(\vec{y})$, $e' = \mathtt{true}$ and
$$\langle \underbrace{\overline{u}^{\mathsf{n}}(\vec{y}).P}_{=S}, \sigma\rangle \xrightarrow{\overline{u}^{\mathsf{n}}(\vec{y})} \langle P, \underbrace{\sigma[\vec{y} \mapsto u]}_{=\sigma'}\rangle \qquad \text{with } S' = P$$
under the hypothesis that $\vec{y} \cap \mathrm{dom}(\sigma) = \emptyset$. By inspecting the typing rules in Figure 10 (page 29), S can be typed only by using rule [VReq], whose hypothesis yields
$$\Delta(u) \equiv \mathcal{G}(\vec{y}) \qquad e \frown \Gamma \vdash P \triangleright \Delta, (\vec{y},0) : \mathbb{T} \qquad \partial\big(\mathsf{nf}(\mathbb{T})\big) = \partial\big(\mathsf{nf}(\mathcal{G}(\vec{y})\restriction 0)\big) \tag{A.1}$$

We show (3) taking $\Delta' = \Delta, (\vec{\mathsf{y}},0) : \mathbb{T}$. We have that

$$\Delta \xrightarrow{\overline{u}^{\mathsf{n}}(\vec{\mathsf{y}})} \Delta' \qquad \text{and} \qquad e \wedge e' \lrcorner \Gamma \vdash S' \triangleright \Delta'$$

respectively by rule [TReq] and by the judgment in (A.1) observing that $e' = \texttt{true}$ and $S' = P$. It remains to prove that $\sigma' \ltimes (e \wedge e'; \Gamma; S'; \Delta')$:

(1) We have $\mathrm{dom}(\Gamma) \subseteq \mathrm{dom}(\sigma) \subseteq \mathrm{dom}(\sigma) \cup \vec{\mathsf{y}} = \mathrm{dom}(\sigma')$ where the first inclusion is implied by $\sigma \ltimes (e; \Gamma; S; \Delta)$. By definition, $\mathrm{dom}(\Delta') = \mathrm{dom}(\Delta) \cup \vec{\mathsf{y}}$, hence $\mathrm{dom}(\Delta'|_{\mathbb{Y}}) = (\mathrm{dom}(\Delta|_{\mathbb{Y}})) \cup \vec{\mathsf{y}}$. Moreover, $\sigma \ltimes (e; \Gamma; S; \Delta)$ implies $(\mathrm{dom}(\Delta|_{\mathbb{Y}})) \subseteq \mathrm{dom}(\sigma)$. Consequently,

$$(\mathrm{dom}(\Delta'|_{\mathbb{Y}})) = (\mathrm{dom}(\Delta|_{\mathbb{Y}})) \cup \vec{\mathsf{y}} \subseteq \mathrm{dom}(\sigma) \cup \vec{\mathsf{y}} = \mathrm{dom}(\sigma')$$

(2) By definition of σ', $\sigma'|_{\mathbb{X}} = \sigma|_{\mathbb{X}}$, hence $\vdash \sigma'(x) : \Gamma(x)$ for all $x \in \mathrm{dom}(\Gamma)$ since $\sigma \ltimes (e; \Gamma; S; \Delta)$ implies $\vdash \sigma(x) : \Gamma(x)$ for all $x \in \mathrm{dom}(\Gamma)$.

(3) From $\sigma'|_{\mathbb{X}} = \sigma|_{\mathbb{X}}$ and $\sigma \ltimes (e; \Gamma; S; \Delta)$, which implies $e \downarrow \sigma = \texttt{true}$ by (3) in Definition 8.1; we have $e \wedge e' \downarrow \sigma' = \texttt{true}$.

(4) Note that $\sigma \ltimes \mathbb{T}$ implies $\sigma' \ltimes \mathbb{T}$ because $\sigma'|_{\mathbb{X}} = \sigma|_{\mathbb{X}}$. Therefore, for all $(\vec{\mathsf{y}},\mathsf{p}) \in \mathrm{dom}(\Delta)$, we have that $\sigma' \ltimes \Delta(\vec{\mathsf{y}},\mathsf{p})$ because $\sigma \ltimes \Delta(\vec{\mathsf{y}},\mathsf{p})$ (which is ensured by $\sigma \ltimes (e; \Gamma; S; \Delta)$). Finally, $\sigma' \ltimes \Delta'(\vec{\mathsf{y}},0)$ follows from Lemma A.19.

- [SAcc] Analogous to the previous case.
- [SRcv] We have

$$\langle \underbrace{\sum_{i \in I} \mathsf{y}_i(x_i).P_i}_{=S}, \sigma \rangle \xrightarrow{\mathsf{y}_j \mathsf{v}} \langle P_j, \underbrace{\sigma[x_j \mapsto \mathsf{v}]}_{=\sigma'} \rangle \qquad \text{where} \qquad S' = P_j$$

with $\alpha = \mathsf{y}_j \mathsf{v}$ for some $j \in I$. As above $e' = \texttt{true}$ and, by typing rule [VRcv], we have

$$\Delta = \Delta'', (\vec{\mathsf{y}},\mathsf{p}) : \sum_{i \in I} e \prec \mathsf{y}_i\ \mathsf{d}_i.\mathbb{T}_i \qquad \forall i \in I : \ e \lrcorner \Gamma, x_i : \mathsf{d}_i \vdash P_i \triangleright \Delta'_i \tag{A.2}$$

where $\Delta'_i = \Delta'', (\vec{\mathsf{y}},\mathsf{p}) : \mathbb{T}_i$ for each $i \in I$. We show (1) taking $\Delta' = \Delta'_j$. From rule [TRcv], we have $\Delta \xrightarrow{\mathsf{y}_j \mathsf{d}_j} \Delta'$. In addition, if $\vdash \mathsf{v} : \mathsf{d}_j$ then take $x = x_j$ and $\mathsf{d} = \mathsf{d}_j$. Judgment $e \wedge e' \lrcorner \Gamma \vdash S' \triangleright \Delta'$ is derived from the j^{th} judgment in (A.2) by noticing that $e' = \texttt{true}$. Then, observing that $\mathrm{dom}(\sigma') = \mathrm{dom}(\sigma) \cup \{x\}$, we prove that $\sigma' \ltimes (e \wedge e'; \Gamma; S'; \Delta')$ as follows:

(1) We have $\mathrm{dom}(\Gamma, x : \mathsf{d}) = \mathrm{dom}(\Gamma) \cup \{x\} \subseteq \mathrm{dom}(\sigma) \cup \{x\} = \mathrm{dom}(\sigma')$ since $\sigma \ltimes (e; \Gamma; S; \Delta)$ implies $\mathrm{dom}(\Gamma) \subseteq \mathrm{dom}(\sigma)$. Noting that $\mathrm{dom}(\Delta') = \mathrm{dom}(\Delta)$, we have $\mathrm{dom}(\Delta'|_{\mathbb{Y}}) = \mathrm{dom}(\Delta|_{\mathbb{Y}}) \subseteq \mathrm{dom}(\sigma)$ because $\sigma \ltimes (e; \Gamma; S; \Delta)$.

(2) Let $\Gamma' = \Gamma \cup \{x\}$. If $\vdash \mathsf{v} : \mathsf{d}$, then $\sigma'(x) = \mathsf{v} : \mathsf{d} = \Gamma'(x)$. We conclude that $\sigma(x') = \sigma(x') : \Gamma'(x')$ for all $x' \neq x \in \mathrm{dom}(\Gamma')$ by $\sigma \ltimes (e; \Gamma; S; \Delta)$. If it is not the case that $\vdash \mathsf{v} : \mathsf{d}$, then there is nothing to prove.

(3) Since $e' = \texttt{true}$, $e \wedge e' \downarrow \sigma' = \texttt{true}$ trivially holds and $\sigma \ltimes (e; \Gamma; S; \Delta)$ implies $e \downarrow \sigma = \texttt{true}$. In addition, $\sigma'(x) = \sigma(x)$ for all $x \in \mathrm{dom}(\sigma)$. Hence, $e \downarrow \sigma' = \texttt{true}$, which implies $(e \wedge e') \downarrow \sigma' = \texttt{true}$.

(4) By Lemma A.19 we have the thesis.

- [SSend] In this case $\alpha = \overline{\mathsf{y}}\mathsf{v}$ with $S = \overline{\mathsf{y}}\, e_1$, $e_1 \downarrow \sigma = \mathsf{v}$, $e' = \texttt{true}$, $S' = 0$ and $\sigma' = \sigma$. Hence, S can be typed only by applying rule [VSend], which yields

$$\Gamma \vdash e_1 : \mathsf{d} \qquad \mathsf{y} \in \vec{\mathsf{y}} \qquad \Delta = \Delta'', (\vec{\mathsf{y}},\mathsf{p}) : e \prec \overline{\mathsf{y}}\ \mathsf{d}; e \prec \mathsf{end}$$

where $\Delta''(\vec{y}',\mathsf{p}) = e \prec \mathsf{end}$ for all $(\vec{y}',\mathsf{p}) \in \mathrm{dom}(\Delta'')$. We show (2) taking $\Delta' = \Delta'', (\vec{y},\mathsf{p}) : \mathsf{end}$. Note that, from the hypothesis $\sigma \ltimes (e;\Gamma;S;\Delta)$, we have $\vdash \sigma(x) \colon \Gamma(x)$ for all $x \in \mathrm{dom}(\Gamma)$, hence $\vdash \mathsf{v} \colon \mathsf{d}$. From rule [TSend], $\Delta \xrightarrow{\overline{\mathsf{yd}}} \Delta'$ and, by rule [VEnd], $e \wedge e' \lrcorner \Gamma \vdash 0 \triangleright \Delta'$.

It is straightforward to conclude that $\sigma' \ltimes (e \wedge e';\Gamma;0;\Delta')$ because $\sigma \ltimes (e;\Gamma;S;\Delta)$, $\mathrm{dom}(\Delta') = \mathrm{dom}(\Delta)$, $\sigma' = \sigma$, and $e' = \mathtt{true}$.

- [SInit] We have

$$\langle \underbrace{\overline{u}^{\mathsf{n}}(\vec{y}).P_0 | u^1(\vec{y}).P_1 | \ldots | u^{\mathsf{n}}(\vec{y}).P_n}_{=S}, \sigma\rangle \xrightarrow{\tau} \langle \underbrace{(\nu \vec{y}@\,\mathsf{u})(P_0 | \ldots | P_n | \vec{y} : \emptyset)}_{=S'}, \underbrace{\sigma[\vec{y} \mapsto u]}_{=\sigma'}\rangle$$

The judgment $e \lrcorner \Gamma \vdash S \triangleright \Delta$ is obtained by repeated application of rule [VPar] on

$$\frac{\Delta_0(u) \equiv \mathcal{G}(\vec{y}) \qquad e \lrcorner \Gamma \vdash P_0 \triangleright \Delta_0, (\vec{y},0) : \mathbb{T}_0 \qquad \partial(\mathbb{T}_0) = \partial(\mathcal{G}(\vec{y})\upharpoonright 0)}{e \lrcorner \Gamma \vdash \overline{u}^{\mathsf{n}}(\vec{y}).P_0 \triangleright \Delta_0} \quad [\mathsf{VReq}]$$

and

$$\frac{\Delta_p(u) \equiv \mathcal{G}(\vec{y}) \qquad e \lrcorner \Gamma \vdash P_p \triangleright \Delta_p, (\vec{y},\mathsf{p}) : \mathbb{T}_p \qquad \partial(\mathbb{T}_p) = \partial(\mathcal{G}(\vec{y})\upharpoonright \mathsf{p})}{e \lrcorner \Gamma \vdash u^{\mathsf{p}}(\vec{y}).P_p \triangleright \Delta_p} \quad [\mathsf{VAcc}]$$

with $p \in \{1,\ldots,n\}$, $\Delta = \Delta_0 \cup \Delta_1 \cup \cdots \cup \Delta_n$, under the assumption that $\Delta_1,\ldots,\Delta_n$ are pairwise independent. We show (3) by taking $\Delta' = \Delta$ and observing that, by rule [VNew], we have

$$\frac{e \lrcorner \Gamma \vdash P_0 \mid \ldots \mid P_n \mid \vec{y} : [] \triangleright \Delta, \{(\vec{y},\mathsf{p}) \mapsto \mathbb{T}_p \mid p \in \{0,\ldots,n\}\}}{e \lrcorner \Gamma \vdash S' \triangleright \Delta}$$

with the premiss of the above derivation obtained by rule [VEmpty] and repeated applications of rule [VPar]. To show that $\sigma' \ltimes (e;\Gamma;S';\Delta)$ we note that $\mathrm{dom}(\sigma') = \mathrm{dom}(\sigma) \cup \vec{y}$ by construction and that

$$\mathtt{fy}(S') = \bigcup_{p=0}^{n} \mathtt{fy}(P_p) \setminus \vec{y} = \bigcup_{p=0}^{n} (\mathtt{fx}(P_p) \setminus \vec{y}) = \mathtt{fy}(S)$$

therefore conditions (1), and (4) of Definition 8.1 hold by inductive hypothesis while conditions (2) and (3) hold because $\sigma'|_{\mathbb{X}} = \sigma|_{\mathbb{X}}$.

- [SForEnd] We have $\ell \downarrow \sigma = \varepsilon$ and

$$\langle \underbrace{\mathsf{for}\ x\ \mathsf{in}\ \ell\ \mathsf{do}\ P}_{=S}, \sigma\rangle \xrightarrow{\tau} \langle 0, \sigma\rangle \qquad \text{with } S' = 0,\ \sigma' = \sigma,\ \text{and } e' = \mathtt{true}$$

Since $e \downarrow \sigma = \mathtt{true}$ (because $\sigma \ltimes (e;\Gamma;S;\Delta)$) we have $e \wedge \ell = \varepsilon \not\vdash \bot$ and, consequently, [VForEnd] is the only applicable rule to type S. Therefore we have $\Delta = \Delta'^{\star}$ for an end-only Δ' (by the premiss of [VForEnd]). Finally, we have that (3) holds by rule [TLoop$_1$] and $\sigma \ltimes (e;\Gamma;0;\Delta')$ directly follows by $\sigma \ltimes (e;\Gamma;S;\Delta)$.

Inductive step.

- [SThen] In this case

$$\frac{e' \downarrow \sigma = \texttt{true} \qquad \langle P_1, \sigma\rangle \xrightarrow{e_1 \vdash \alpha} \langle P_1', \sigma'\rangle}{\langle \underbrace{\mathsf{if}\ e'\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2}_{=S}, \sigma\rangle \xrightarrow{e' \wedge e_1 \vdash \alpha} \langle P_1', \sigma'\rangle}$$

  with $S' = P_1'$. To type S we must apply rule [VIf]

$$\frac{e \wedge e' \lrcorner \Gamma \ \vdash\ P_1 \ \triangleright\ \Delta_1 \qquad e \wedge \neg e' \lrcorner \Gamma \ \vdash\ P_2 \ \triangleright\ \Delta_2}{e \lrcorner \Gamma \ \vdash\ \mathsf{if}\ e'\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2 \ \triangleright\ \underbrace{\Delta_1 \bowtie \Delta_2}_{=\Delta}}\ [\mathsf{VIf}]$$

  recall that $\Delta_1 \bowtie \Delta_2$ is defined when Δ_1 and Δ_2 are mergeable (i.e., $\mathrm{dom}(\Delta_1) = \mathrm{dom}(\Delta_2)$ and the pseudo-types $\Delta_1(\vec{\mathsf{y}},\mathsf{p})$ and $\Delta_2(\vec{\mathsf{y}},\mathsf{p})$ are mergeable for all $(\vec{\mathsf{y}},\mathsf{p}) \in \mathrm{dom}(\Delta_1)$).

  By the inductive hypothesis, there exist Γ' and Δ_1' such that

  (1) if $\alpha = \mathsf{y}\mathsf{v}$ then $\Delta_1 \overset{\mathsf{yd}}{\Longrightarrow} \Delta_1'$; moreover, if $\vdash \mathsf{v}\colon \mathsf{d}$ then

$$e \wedge e_1 \wedge e' \lrcorner \Gamma', x : \mathsf{d} \ \vdash\ P_1' \ \triangleright\ \Delta_1'$$

  and $\sigma' \ltimes (e \wedge e_1 \wedge e'; \Gamma', x : \mathsf{d}; P_1'; \Delta_1')$. By Lemma B.2, $\Delta = \Delta_1 \bowtie \Delta_2 \overset{\mathsf{yd}}{\Longrightarrow} \Delta_1'$. Then, take $\Delta' = \Delta_1'$ and note that

$$e \wedge (e_1 \wedge e') \lrcorner \Gamma', x : \mathsf{d} \ \vdash\ P_1' \ \triangleright\ \Delta'$$

  and $\sigma' \ltimes (e \wedge (e' \wedge e'); \Gamma, x : \mathsf{d}; P_1'; \Delta')$ hold by associativity of $\wedge$.

  (2) and (3) analogous to the previous case.

- [SElse] Analogous to [SThen].
- [SSeq] We have

$$\frac{\langle P_1, \sigma\rangle \xrightarrow{e' \vdash \alpha} \langle P_1', \sigma'\rangle}{\langle \underbrace{P_1; P_2}_{=S}, \sigma\rangle \xrightarrow{e' \vdash \alpha} \langle \underbrace{P_1'; P_2}_{=S'}, \sigma'\rangle}$$

  As in the previous case, we can type S only applying [VSeq]. This implies that $\Delta = \Delta_1; \Delta_2$ for some Δ_1 and Δ_2 such that

  – $\mathrm{dom}(\Delta_2) \subseteq \mathrm{dom}(\Delta_1)$ and $\Delta_1|_{\mathbb{U}\cup\mathbb{Y}} = \Delta_2|_{\mathbb{U}\cup\mathbb{Y}} = \Delta|_{\mathbb{U}\cup\mathbb{Y}}$

  – $\Delta : (\vec{\mathsf{y}},\mathsf{p}) \mapsto \begin{cases} \Delta_1(\vec{\mathsf{y}},\mathsf{p}); \Delta_2(\vec{\mathsf{y}},\mathsf{p}) & (\vec{\mathsf{y}},\mathsf{p}) \in \mathrm{dom}(\Delta_2) \\ \Delta_1(\vec{\mathsf{y}},\mathsf{p}) & (\vec{\mathsf{y}},\mathsf{p}) \in \mathrm{dom}(\Delta_1) \setminus \mathrm{dom}(\Delta_2) \\ \text{undef} & \text{otherwise} \end{cases}$

  – $e \lrcorner \Gamma \ \vdash\ P_1 \ \triangleright\ \Delta_1$ and $e \lrcorner \Gamma \ \vdash\ P_2 \ \triangleright\ \Delta_2$.

  By the inductive hypothesis, there are Γ' and Δ_1' such that

  (1) if $\alpha = \mathsf{y}\mathsf{v}$ then $\Delta_1 \xrightarrow{\mathsf{yd}} \Delta_1'$ for a sort d; also, if $\vdash \mathsf{v}\colon \mathsf{d}$ then there is $x \in \mathbb{X}$ such that $\sigma'(x) = \mathsf{v}$, $e \wedge e' \lrcorner \Gamma', x : \mathsf{d} \ \vdash\ P_1' \ \triangleright\ \Delta_1'$ and $\sigma' \ltimes (e \wedge e'; \Gamma', x : \mathsf{d}; P_1'; \Delta_1')$

  (2) if $\alpha = \overline{\mathsf{y}}\mathsf{v}$ then $\Delta_1 \xrightarrow{\overline{\mathsf{y}}\mathsf{d}} \Delta_1'$ with $\vdash \mathsf{v}\colon \mathsf{d}$, $\sigma' \ltimes (e \wedge e'; \Gamma'; P_1'; \Delta_1')$, and $e \wedge e' \lrcorner \Gamma' \ \vdash\ P_1' \triangleright \Delta_1'$

  (3) otherwise $\Delta_1 \xrightarrow{\alpha} \Delta_1'$, $\sigma' \ltimes (e \wedge e'; \Gamma'; P_1'; \Delta_1')$, and $e \wedge e' \lrcorner \Gamma' \ \vdash\ P_1' \ \triangleright\ \Delta_1'$

  In any case, by rule [TSeq] we have that

$$\Delta = \Delta_1; \Delta_2 \xrightarrow{\alpha} \Delta_1'; \Delta_2 = \Delta'$$

and the proof concludes with the application of rule [VSeq] to the judgement typing P_1' and $e \lrcorner \Gamma' \vdash P_2 \triangleright \Delta_2$ observing that $\mathrm{dom}(\Gamma') \subseteq \mathrm{dom}(\Gamma)$ and $\mathrm{dom}(\Delta_2) \subseteq \mathrm{dom}(\Delta_1) \subseteq \mathrm{dom}(\Delta_1')$ where the latter inclusion holds because the domain of a specification may only grow after transitions (by inspection of the rules in Figure 12) and, for the same reason, $\Delta_2|_{\mathbb{U}\cup\mathbb{Y}} = \Delta_1|_{\mathbb{U}\cup\mathbb{Y}} = \Delta_1'|_{\mathbb{U}\cup\mathbb{Y}}$ since Δ_1' and Δ_1 differ at one participant's session only.

- [SFor] We have

$$\frac{\ell \downarrow \sigma \neq \varepsilon \qquad \langle P, \sigma[x \mapsto \mathsf{hd}(\ell \downarrow \sigma)]\rangle \xrightarrow{e' \vdash \alpha} \langle P', \sigma'\rangle}{\langle \underbrace{\mathsf{for}\ x\ \mathsf{in}\ \ell\ \mathsf{do}\ P}_{=S}, \sigma\rangle \xrightarrow{e' \vdash \alpha} \langle \underbrace{P'; \mathsf{for}\ x\ \mathsf{in}\ \mathsf{tl}(\ell)\ \mathsf{do}\ P}_{S'}, \sigma'\rangle} \tag{A.3}$$

The first premiss of (A.3) implies $e \wedge \ell \neq \varepsilon \not\Rightarrow \bot$, hence the only applicable rule to type S is [VFor] which we instantiate as:

$$\frac{\Gamma \vdash \ell : [\mathsf{d}] \qquad e \lrcorner \Gamma, x : \mathsf{d} \vdash P \triangleright \Delta_1 \qquad \Delta_1 \text{ active} \qquad x \notin \mathrm{var}(\Delta_1)}{e \lrcorner \Gamma \vdash \mathsf{for}\ x\ \mathsf{in}\ \ell\ \mathsf{do}\ P \triangleright \underbrace{\Delta_1^*}_{=\Delta}} \tag{A.4}$$

for some Δ_1. In order to use the inductive hypothesis, we check that

$$\sigma[x \mapsto \mathsf{hd}(\ell \downarrow \sigma)] \ltimes (e; \Gamma, x : \mathsf{d}; P; \Delta_1)$$

In fact, from $\sigma \ltimes (e; \Gamma; S; \Delta)$ we have that

(1) $\mathrm{dom}(\Gamma, x : \mathsf{d}) = \mathrm{dom}(\Gamma) \cup \{x\} \subseteq \mathrm{dom}(\sigma) \cup \{x\} = \mathrm{dom}(\sigma[x \mapsto \mathsf{hd}(\ell \downarrow \sigma)])$; also, $\mathrm{dom}(\Delta_1|_{\mathbb{Y}}) \subseteq \mathrm{dom}(\sigma) \subseteq \mathrm{dom}(\sigma[x \mapsto \mathsf{hd}(\ell \downarrow \sigma)])$

(2) for all $z \in \mathrm{dom}(\Gamma)$ we have $\vdash \sigma[x \mapsto \mathsf{hd}(\ell \downarrow \sigma)](z) : \Gamma(z)$ since $\sigma[x \mapsto \mathsf{hd}(\ell \downarrow \sigma)](z) = \sigma(z)$; moreover, $\Gamma \vdash \ell : [\mathsf{d}]$ implies that $\Gamma \vdash \mathsf{hd}(\ell \downarrow \sigma) : \mathsf{d}$ and therefore $\mathsf{hd}(\ell \downarrow \sigma) = \sigma[x \mapsto \mathsf{hd}(\ell \downarrow \sigma)](x)$ has sort d

(3) since $e \downarrow \sigma = \mathtt{true}$, $x \notin \mathrm{var}(e)$ and therefore $e \downarrow \sigma[x \mapsto \mathsf{hd}(\ell \downarrow \sigma)] = \mathtt{true}$

(4) $\forall(\vec{\mathsf{y}},\mathsf{p}) \in \mathrm{dom}(\Delta_1), \sigma[x \mapsto \mathsf{hd}(\ell \downarrow \sigma)] \ltimes \Delta_1(\vec{\mathsf{y}},\mathsf{p})$ follows from Lemma A.19.

We proceed by case analysis on α. First note that By Lemma A.20, we can exclude the case $\alpha = \mathsf{y}\mathsf{v}$ because Δ_1 is active. We consider the case $\alpha = \overline{\mathsf{y}}\mathsf{v}$ (the proof in the other cases is analogous and simpler and therefore omitted). Assume that $\vdash \mathsf{v} : \mathsf{d}'$. By the inductive hypothesis on (A.3) and (A.4), there exists $\Delta_1 \xrightarrow{\overline{\mathsf{y}}\mathsf{d}'} \Delta_1'$ such that

$$e \wedge e' \lrcorner \Gamma, x : \mathsf{d} \vdash P' \triangleright \Delta_1' \quad \text{with } \sigma' \ltimes (e \wedge e'; \Gamma, x : \mathsf{d}; P'; \Delta_1') \tag{A.5}$$

We show how to find an environment Δ' required in (2) depending on the possible typings of $\mathsf{for}\ x\ \mathsf{in}\ \mathsf{tl}(\ell)\ \mathsf{do}\ P$.

If $e \wedge e' \wedge \mathsf{tl}(\ell) = \varepsilon \not\Rightarrow \bot$ then, using rule [VForEnd], we have

$$e \wedge e' \lrcorner \Gamma, x : \mathsf{d} \vdash \mathsf{for}\ x\ \mathsf{in}\ \mathsf{tl}(\ell)\ \mathsf{do}\ P \triangleright \Delta_1'|_{\mathbb{U}\cup\mathbb{Y}} \tag{A.6}$$

since $\forall(\vec{\mathsf{y}},\mathsf{p}) \in \mathrm{dom}(\Delta_1'|_{\mathbb{U}\cup\mathbb{Y}}) : (\Delta_1'|_{\mathbb{U}\cup\mathbb{Y}})(\vec{\mathsf{y}},\mathsf{p}) = \mathsf{end}$ vacuously holds. Then, from (A.5) and (A.6), we derive $e \wedge e' \lrcorner \Gamma, x : \mathsf{d} \vdash S' \triangleright \Delta_1'; (\Delta_1'|_{\mathbb{U}\cup\mathbb{Y}})$ by using rule [VSeq]. Finally, $\Delta = \Delta_1^\star \xrightarrow{\overline{\mathsf{y}}\mathsf{d}'} \Delta_1'$ by [TLoop$_1$] and the thesis follows by observing that $\Delta_1' = \Delta_1'; (\Delta_1'|_{\mathbb{U}\cup\mathbb{Y}})$ since $\mathrm{dom}(\Delta_1') \cap \mathrm{dom}(\Delta_1'; (\Delta_1'|_{\mathbb{U}\cup\mathbb{Y}})) \subseteq \mathbb{U} \cup \mathbb{Y}$ by definition of $\_;\_$ on specifications (*cf.* page 26).

If $e \wedge e' \wedge \mathsf{tl}(\ell) \neq \varepsilon \not\forall \bot$ then, using rule [VFor],

$$\frac{\Gamma \vdash \mathsf{tl}(\ell) \colon [\mathsf{d}] \qquad e \frown \Gamma, x : \mathsf{d} \vdash P \triangleright \Delta_1 \qquad \Delta_1 \text{ active} \qquad x \notin \mathrm{var}(\Delta_1)}{e \frown \Gamma, x : \mathsf{d} \vdash \mathsf{for}\ x\ \mathsf{in}\ \mathsf{tl}(\ell)\ \mathsf{do}\ P \triangleright \Delta_1^*} \tag{A.7}$$

where $\Gamma \vdash \mathsf{tl}(\ell) \colon [\mathsf{d}]$ holds because $\Gamma \vdash \ell \colon [\mathsf{d}]$ and the conditions in the rest of the hypothesis of (A.7) hold by the typing (A.4). By Lemma A.14 and Lemma A.10 on the conclusion, from (A.7)

$$e \wedge e' \frown \Gamma, x : \mathsf{d} \vdash \mathsf{for}\ x\ \mathsf{in}\ \mathsf{tl}(\ell)\ \mathsf{do}\ P \triangleright \Delta'' \tag{A.8}$$

where $\Delta'' = \Delta_1^\star[(\vec{\mathsf{y}},\mathsf{p}) \mapsto e' \prec \mathsf{end}; \Delta_1^\star(\vec{\mathsf{y}},\mathsf{p})]_{(\vec{\mathsf{y}},\mathsf{p}) \in \mathrm{dom}(\Delta_1^\star)}$.

We now show that $\Delta_1'; \Delta''$ and $\Delta_1'; \Delta$ are defined and equivalent. Note that $\mathrm{dom}(\Delta'') = \mathrm{dom}(\Delta) = \mathrm{dom}(\Delta_1) \subseteq \mathrm{dom}(\Delta_1')$ because $\Delta_1 \xrightarrow{\bar{\mathsf{y}}\mathsf{d}} \Delta_1'$. For the equivalence, we just need to consider session names; let $(\vec{\mathsf{y}},\mathsf{p}) \in \mathrm{dom}(\Delta_1'; \Delta'')$, we have two cases: if $(\vec{\mathsf{y}},\mathsf{p}) \notin \mathrm{dom}(\Delta'')$ then $(\Delta_1'; \Delta'')(\vec{\mathsf{y}},\mathsf{p}) = \Delta_1'(\vec{\mathsf{y}},\mathsf{p}) = (\Delta_1'; \Delta)(\vec{\mathsf{y}},\mathsf{p})$ by the definition of $\_;\_$ and if $(\vec{\mathsf{y}},\mathsf{p}) \in \mathrm{dom}(\Delta'')$ then

$$\begin{array}{rcll} (\Delta_1'; \Delta'')(\vec{\mathsf{y}},\mathsf{p}) = & \Delta_1'(\vec{\mathsf{y}},\mathsf{p}); \Delta''(\vec{\mathsf{y}},\mathsf{p}); & & \text{by def. of } \_;\_ \\ = & \Delta_1'(\vec{\mathsf{y}},\mathsf{p}); (e' \prec \mathsf{end}; \Delta(\vec{\mathsf{y}},\mathsf{p})); & & \text{by def. of } \Delta'' \\ = & \Delta_1'(\vec{\mathsf{y}},\mathsf{p}); \Delta(\vec{\mathsf{y}},\mathsf{p}); & & \text{by Lemma A.7} \end{array}$$

By rule [VSeq], with the judgments in (A.5) and (A.8), we have

$$e \wedge e' \frown \Gamma, x : \mathsf{d} \vdash S' \triangleright \Delta_1'; \Delta \tag{A.9}$$

and $\Delta = \Delta_1^\star \xrightarrow{\mathsf{yd}} \Delta_1'; \Delta = \Delta'$ by rule [TLoop$_2$].

It remains to show that $\sigma' \ltimes (e \wedge e'; \Gamma, x : \mathsf{d}; S'; \Delta')$ holds:

(1) It follows immediately from (A.5) and the fact that $\mathrm{dom}(\Delta') = \mathrm{dom}(\Delta_1')$.
(2) Trivially from (A.5).
(3) Immediate from (A.5);
(4) By Lemma A.19.

- [SLoopEnd] We have

$$\frac{\langle M, \sigma\rangle \xrightarrow{e' \vdash \alpha} \langle P, \sigma'\rangle}{\langle \underbrace{\mathsf{repeat}\ N\ \mathsf{until}\ M}_{=S}, \sigma\rangle \xrightarrow{e' \vdash \alpha} \langle P, \sigma'\rangle} \qquad \text{with } S' = P$$

and the only applicable rule to type S is [VLoop]:

$$\frac{e \frown \Gamma \vdash N \triangleright \Delta_1 \qquad e \frown \Gamma \vdash M \triangleright \Delta_2 \qquad \Delta_1 \text{ and } \Delta_2 \text{ passively compatible}}{e \frown \Gamma \vdash \mathsf{repeat}\ N\ \mathsf{until}\ M \triangleright \Delta_1^\star; \Delta_2}$$

with $\Delta = \Delta_1^\star; \Delta_2$. The inductive hypothesis requires to have $\sigma \ltimes (e; \Gamma; M; \Delta_2)$, which easily follows because (1)-(3) are immediate from $\sigma \ltimes (e; \Gamma; S; \Delta)$ and (4) follows from Lemma A.19. Note that α has to be an input label yv because M is a choice process. Hence, there exist Γ' and $\Delta_2 \xrightarrow{\mathsf{yd}} \Delta_2'$ and if $\vdash \mathsf{v} \colon \mathsf{d}$ then

$$e \wedge e' \frown \Gamma', x : \mathsf{d} \vdash P \triangleright \Delta_2' \qquad \text{and} \qquad \sigma' \ltimes (e \wedge e'; \Gamma', x : \mathsf{d}; P; \Delta_2')$$

Then, take $\Delta' = \Delta_2'$ and note that by [TLoop$_0$] $\Delta = \Delta_1^\star; \Delta_2 \xrightarrow{\mathsf{yd}} \Delta'$.

- [SLoop] We have

$$\frac{\langle N,\sigma\rangle \xrightarrow{e'\vdash\alpha} \langle P,\sigma'\rangle \quad M=\sum_{i\in I}\mathsf{y}_i(x_i).P_i \quad \forall i\in I.\mathsf{y}_i\notin \mathtt{fy}(\alpha)}{\langle \underbrace{\mathsf{repeat}\ N\ \mathsf{until}\ M}_{=S},\sigma\rangle \xrightarrow{e'\vdash\alpha} \langle \underbrace{P;\mathsf{repeat}\ N\ \mathsf{until}\ M}_{=S'},\sigma'\rangle} \tag{A.10}$$

and, as in the case [SLoopEnd], the only applicable rule to type S is [VLoop]:

$$\frac{e\lrcorner\Gamma \vdash N \triangleright \Delta_1 \qquad e\lrcorner\Gamma \vdash M \triangleright \Delta_2 \qquad \Delta_1 \text{ and } \Delta_2 \text{ passively compatible}}{e\lrcorner\Gamma \vdash \mathsf{repeat}\ N\ \mathsf{until}\ M \triangleright \Delta_1^\star;\Delta_2}$$

with $\Delta=\Delta_1^\star;\Delta_2$. The inductive hypothesis requires to show $\sigma\ltimes(e;\Gamma;N;\Delta_1)$, which holds because (1)-(3) are immediate from $\sigma\ltimes(e;\Gamma;S;\Delta)$ and (4) follows from Lemma A.19. Note that α has to be an input label $\mathsf{y}\mathsf{v}$ because N is a choice process. Hence, there exist Γ' and $\Delta_1\xrightarrow{\mathsf{y}\mathsf{d}}\Delta_1'$ and if $\vdash \mathsf{v}:\mathsf{d}$ then

$$e\wedge e'\lrcorner\Gamma',x:\mathsf{d}\vdash P\triangleright\Delta_1' \qquad \text{and} \qquad \sigma'\ltimes(e\wedge e';\Gamma',x:\mathsf{d};P;\Delta_1')$$

Then, take $\Delta'=\Delta_1';\Delta_1^\star$ and note that by [TLoop$_2$] $\Delta=\Delta_1^\star;\Delta_2\xrightarrow{\mathsf{y}\mathsf{d}}\Delta'$. Finally, the thesis follows by applying [VSeq].

- [SCom$_1$] In this case:

$$\frac{\langle P,\sigma\rangle \xrightarrow{e'\vdash \overline{\mathsf{y}}\mathsf{v}} \langle P',\sigma'\rangle}{\langle \underbrace{P|\mathsf{y}[\vec{\mathsf{v}}]}_{=S},\sigma\rangle \xrightarrow{e'\vdash\tau} \langle \underbrace{P'|\mathsf{y}[\vec{\mathsf{v}}\cdot\mathsf{v}]}_{=S'},\sigma'\rangle}$$

and, by rule [VPar], we have the following typing for S:

$$\frac{e\lrcorner\Gamma \vdash P \triangleright \overbrace{\Delta\setminus\{\mathsf{y}:\vec{\mathsf{d}}\}}^{=\Delta_1} \qquad e\lrcorner\Gamma\vdash \mathsf{y}[\vec{\mathsf{v}}]\triangleright \overbrace{\Delta|_{\mathbb{U}},\mathsf{y}:\vec{\mathsf{d}}}^{=\Delta_2} \qquad \Delta_1 \text{ and } \Delta_2 \text{ independent}}{e\lrcorner\Gamma\vdash S\triangleright\Delta}$$

where the typing of the queue $\mathsf{y}[\vec{\mathsf{v}}]$ is obtained by repeated applications of the rule [VQueue] ending with an application of [VEmpty] and Lemma A.10; hence, $\vdash\vec{\mathsf{v}}:\vec{\mathsf{d}}$. Note that $\Delta=\Delta_1\cup\Delta_2$. We now observe that $\sigma\ltimes(e;\Gamma;P;\Delta_1)$ and $\sigma\ltimes(e;\Gamma;P;\Delta_2)$ directly follow from $\sigma\ltimes(e;\Gamma;S;\Delta)$ (since $\mathrm{dom}(\Delta_1)\cup\mathrm{dom}(\Delta_2)\subseteq\mathrm{dom}(\Delta)$) so, by inductive hypothesis,

$$\Delta_1\xrightarrow{\overline{\mathsf{y}}\mathsf{d}}\Delta_1' \quad\text{with}\quad \vdash\mathsf{v}:\mathsf{d} \quad e\wedge e'\lrcorner\Gamma\vdash P'\triangleright\Delta_1' \quad\text{and}\quad \sigma'\ltimes(e\wedge e';\Gamma';P';\Delta_1')$$

for some environment Γ'. We now show that we can type S' with $\Delta'=\Delta_1'\cup\{\mathsf{y}:\vec{\mathsf{d}}\cdot\mathsf{d}\}$:

$$\frac{e\wedge e'\lrcorner\Gamma'\vdash P'\triangleright\Delta_1' \qquad \dfrac{\vdash\mathsf{v}:\mathsf{d} \qquad e\wedge e'\lrcorner\Gamma'\vdash\mathsf{y}[\vec{\mathsf{v}}]\triangleright\mathsf{y}:[\vec{\mathsf{d}}]}{e\wedge e'\lrcorner\Gamma'\vdash\mathsf{y}[\vec{\mathsf{v}}\cdot\mathsf{v}]\triangleright\mathsf{y}:\vec{\mathsf{d}}\cdot\mathsf{d}}\ [\mathsf{VQueue}]}{e\wedge e'\lrcorner\Gamma'\vdash S'\triangleright\Delta'}\ [\mathsf{VPar}]$$

where the judgement in the premiss of [VQueue] holds by Lemma A.10 and the application of [VPar] is possible because Δ_1' and $\mathsf{y}:\vec{\mathsf{d}}\cdot\mathsf{d}$ are independent. The proof of this case ends by showing that $\sigma'\ltimes(e\wedge e';\Gamma';P';\Delta')$: we have that (1) of Definition 8.1 holds because $\mathtt{fy}(S')=\mathtt{fy}(P')\cup\{\mathsf{y}\}$ while $\mathrm{dom}(\Delta'|_{\mathbb{Y}})=\mathrm{dom}(\Delta_1'|_{\mathbb{Y}})\cup\{\mathsf{y}\}$ by definition and, by Lemma A.23, $\mathrm{dom}(\sigma)\subseteq\mathrm{dom}(\sigma')$; also, conditions (2) and (3) hold because $\sigma'|_{\mathbb{X}}=\sigma|_{\mathbb{X}}$ while (4) holds because for each $(\vec{\mathsf{y}},\mathsf{p})$ we have $\Delta'(\vec{\mathsf{y}},\mathsf{p})=\Delta(\vec{\mathsf{y}},\mathsf{p})$ by construction.

- [SCom$_2$] In this case:

$$\frac{\langle P,\sigma\rangle \xrightarrow{e'\vdash \mathsf{y}\mathsf{v}} \langle P',\sigma'\rangle}{\langle \underbrace{P|\mathsf{y}[\mathsf{v}\cdot\vec{\mathsf{v}}]}_{=S},\sigma\rangle \xrightarrow{e'\vdash\tau} \langle \underbrace{P'|\mathsf{y}[\vec{\mathsf{v}}]}_{=S'},\sigma'\rangle}$$

and, assuming $\Delta_1 = \Delta \setminus \{\mathsf{y} : \mathsf{d}\cdot\vec{\mathsf{d}}\}$, we observe that $\sigma \ltimes (e;\Gamma;P;\Delta_1)$ directly follows from $\sigma \ltimes (e;\Gamma;S;\Delta)$ (since $\mathrm{dom}(\Delta_1) \subseteq \mathrm{dom}(\Delta)$) so, by inductive hypothesis. We can type S as follows

$$\frac{e \lrcorner \Gamma \vdash P \triangleright \Delta_1 \qquad e \lrcorner \Gamma \vdash \mathsf{y}[\mathsf{v}\cdot\vec{\mathsf{v}}] \triangleright \Delta|_{\mathbb{U}}, \mathsf{y} : \mathsf{d}\cdot\vec{\mathsf{d}}}{e \lrcorner \Gamma \vdash S \triangleright \Delta}$$

by using rule [VPar] since Δ_1 and $\Delta|_{\mathbb{U}}, \mathsf{y} : \mathsf{d}\cdot\vec{\mathsf{d}}$ are independent and the typing of the queue $\mathsf{y}[\mathsf{v}\cdot\vec{\mathsf{v}}]$ is obtained by repeated applications of the rule [VQueue] ending with an application of [VEmpty], which yields $\vdash \vec{\mathsf{v}} : \mathsf{d}$ and $\vdash \vec{\mathsf{v}} : \vec{\mathsf{d}}$. By the inductive hypothesis $\Delta_1 \xrightarrow{\mathsf{y}\mathsf{d}} \Delta_1'$ and there are Γ' and $x \in \mathbb{X}$ such that $e \lrcorner \Gamma', x : \mathsf{d} \vdash P' \triangleright \Delta_1'$ and $\sigma' \ltimes (e;\Gamma', x : \mathsf{d};P';\Delta_1')$.

We now show that we can type S' with $\Delta' = \Delta_1' \cup \{\mathsf{y} : \vec{\mathsf{d}}\}$:

$$\frac{e \wedge e' \lrcorner \Gamma', x : \mathsf{d} \vdash P' \triangleright \Delta_1' \qquad e \wedge e' \lrcorner \Gamma', x : \mathsf{d} \vdash \mathsf{y}[\vec{\mathsf{v}}] \triangleright \mathsf{y} : \vec{\mathsf{d}}}{e \wedge e' \lrcorner \Gamma', x : \mathsf{d} \vdash S' \triangleright \Delta'}\ [\mathsf{VPar}]$$

where the second judgment in the premiss holds by Lemma A.10 and the application of [VPar] is possible because Δ_1' and $\mathsf{y} : \vec{\mathsf{d}}$ are independent since $\mathsf{y} \notin \mathrm{dom}(\Delta_1')$ otherwise Δ_1 would not be independent of $\Delta|_{\mathbb{U}}, \mathsf{y} : \vec{\mathsf{d}}$. The proof of this case ends by showing that $\sigma' \ltimes (e \wedge e';\Gamma', x : \mathsf{d};P';\Delta')$. Since $\mathtt{fy}(S') = \mathtt{fy}(P') \cup \{\mathsf{y}\}$, we have that (1) of Definition 8.1 holds; also, $\mathrm{dom}(\Delta'|_{\mathbb{Y}}) = \mathrm{dom}(\Delta_1'|_{\mathbb{Y}}) \cup \{\mathsf{y}\}$ by definition and, by Lemma A.23, $\mathrm{dom}(\sigma) \subseteq \mathrm{dom}(\sigma')$; also, conditions (2) and (3) hold because $\sigma'|_{\mathbb{X}} = \sigma|_{\mathbb{X}}$ and $\sigma'(x) = \mathsf{v}$ has sort $\mathsf{d} = \Gamma'(x)$; finally, (4) holds because for each $(\vec{\mathsf{y}},\mathsf{p})$ we have $\Delta'(\vec{\mathsf{y}},\mathsf{p}) = \Delta(\vec{\mathsf{y}},\mathsf{p})$ by construction.

- [SPar] In this case we have:

$$\frac{\langle S_1,\sigma\rangle \xrightarrow{e'\vdash\alpha} \langle S_1',\sigma'\rangle \quad \mathtt{bn}(\alpha) \cap \mathrm{dom}(\sigma) = \emptyset \quad \mathtt{fx}(S_2) \cap (\mathrm{dom}(\sigma') \setminus \mathrm{dom}(\sigma)) = \emptyset}{\langle \underbrace{S_1|S_2}_{=S},\sigma\rangle \xrightarrow{e'\vdash\alpha} \langle \underbrace{S_1'|S_2}_{=S'},\sigma'\rangle}$$

and the only applicable rule to type S is [VPar]:

$$\frac{e_1 \lrcorner \Gamma \vdash S_1 \triangleright \Delta_1 \qquad e_2 \lrcorner \Gamma \vdash S_2 \triangleright \Delta_2 \qquad \Delta_1 \text{ and } \Delta_2 \text{ independent}}{\underbrace{e_1 \wedge e_2}_{=e} \lrcorner \Gamma \vdash S_1|S_2 \triangleright \underbrace{\Delta_1 \cup \Delta_2}_{=\Delta}}$$

Noting that $\sigma \ltimes (e;\Gamma;S;\Delta)$ trivially implies $\sigma \ltimes (e_1;\Gamma;S_1;\Delta_1)$, by the inductive hypothesis there Γ' and Δ_1' such that

(1) if $\alpha = \mathsf{y}\mathsf{v}$ then $\Delta_1 \xrightarrow{\mathsf{y}\mathsf{d}} \Delta_1'$ for a sort d; also, if $\vdash \mathsf{v} : \mathsf{d}$ then there is $x \in \mathbb{X}$ such that $\sigma'(x) = \mathsf{v}$, $e_1 \wedge e' \lrcorner \Gamma', x : \mathsf{d} \vdash S_1' \triangleright \Delta_1'$ and $\sigma' \ltimes (e_1 \wedge e';\Gamma', x : \mathsf{d};S_1';\Delta_1')$

(2) if $\alpha = \overline{\mathsf{y}}\mathsf{v}$ then $\Delta_1 \xrightarrow{\overline{\mathsf{y}}\mathsf{d}} \Delta_1'$ with $\vdash \mathsf{v} : \mathsf{d}$, $\sigma' \ltimes (e_1 \wedge e';\Gamma;S_1';\Delta_1')$, and $e_1 \wedge e' \lrcorner \Gamma \vdash S_1' \triangleright \Delta_1'$

(3) otherwise $\Delta_1 \xrightarrow{\alpha} \Delta_1'$, $\sigma' \ltimes (e_1 \wedge e';\Gamma';S_1';\Delta_1')$, and $e_1 \wedge e' \lrcorner \Gamma' \vdash S_1' \triangleright \Delta_1'$

In the first two cases Δ_1' and Δ_2 are independent because $\mathrm{dom}(\Delta_1') = \mathrm{dom}(\Delta_1)$ and Δ_1 and Δ_2 are independent. In the latter case, we observe that $\sigma' \ltimes (e_1 \wedge e'; \Gamma'; S_1'; \Delta_1')$ implies $\mathrm{dom}(\Delta_1') \subseteq \mathrm{dom}(\sigma')$; moreover, σ' can contain a new session, say $\vec{y}$ only if the rule [SInit] had been used to derive the transition. In this case, the side condition of [SInit] guarantees that $\vec{y} \notin \mathrm{dom}(\sigma)$ and therefore they do not occur in S_2 and therefore they are not in $\mathrm{dom}(\Delta_2)$ since $\sigma \ltimes (e; \Gamma; S; \Delta)$. The judgement $e_2 \lrcorner \Gamma' \vdash S_2 \triangleright \Delta_2$ holds because condition $\mathtt{fx}(S_2) \cap (\mathrm{dom}(\sigma') \setminus \mathrm{dom}(\sigma)) = \emptyset$ makes $\mathtt{fx}(S_2) \cap \mathrm{dom}(\Gamma') = \emptyset$, hence we apply Lemma B.3.

- [SNew] In this case we have:

$$\frac{\langle S_1, \sigma\rangle \xrightarrow{e' \vdash \alpha} \langle S_1', \sigma'\rangle \qquad \vec{y} \cap \mathtt{fy}(\alpha) = \emptyset}{\langle \underbrace{(\nu \vec{y}@u)S_1}_{=S}, \sigma\rangle \xrightarrow{e' \vdash \alpha} \langle \underbrace{(\nu \vec{y}@u)S_1'}_{=S'}, \sigma'\rangle} \qquad \frac{e \lrcorner \Gamma \vdash S \triangleright \Delta' \qquad \Delta = \Delta'|_{-\vec{y}}}{e \lrcorner \Gamma \vdash (\nu \vec{y}@u)S \triangleright \Delta}$$

  where on the right we have the typing of S obtained by applying rule [VNew]. The thesis immediately follows from the inductive hypothesis since $\mathtt{fy}(S) = \mathtt{fy}(S_1) \setminus \vec{y}$ and $\mathtt{fy}(S') = \mathtt{fy}(S_1') \setminus \vec{y}$.
- [SStr] The thesis is immediate by using the inductive hypothesis. □

Appendix B. WSI by typing

B.1. **Correspondence between semantics of specifications.** The next results show that the denotational semantics of specifications coincides with the operational rules given in Figure 12.

Lemma B.1. *If* $\Delta \xrightarrow{\alpha} \Delta'$ *then* $\Delta, \Delta'' \xrightarrow{\alpha} \Delta', \Delta''$.

Proof. By straightforward induction on the structure of the proof $\Delta \xrightarrow{\alpha} \Delta'$. □

Lemma B.2. $\Delta_1 \xrightarrow{\alpha} \Delta_1'$, *then* $\Delta_1 \bowtie \Delta_2 \xrightarrow{\alpha} \Delta_1'$.

Proof. By induction on the structure of the derivation $\Delta_1 \xrightarrow{\alpha} \Delta_1'$. □

Lemma B.3. $\mathcal{R}_{\vec{y}}(\Delta_1, \Delta_2) = \mathcal{R}_{\vec{y}}(\Delta_1)$ *if* $\mathrm{dom}(\Delta_2) \cap \vec{y} = \emptyset$.

Proof. By straightforward induction on the structure of the proof $r \in \mathcal{R}_{\vec{y}}(\Delta_1, \Delta_2)$. □

Lemma B.4. *Let* Δ *be a specification and* $\vec{y} \in \mathrm{dom}(\Delta)$. *If* $\Delta \xrightarrow{\alpha} \Delta'$ *and* $\mathtt{n}(\alpha) \cap \vec{y} = \emptyset$, *then* $\Delta(\vec{y},\mathsf{p}) = \Delta'(\vec{y},\mathsf{p})$ *for all* $(\vec{y},\mathsf{p}) \in \Delta$.

Proof. By straightforward induction on the structure of the proof $\Delta \xrightarrow{\alpha} \Delta'$. □

Lemma 8.5. *Let* Δ *be a specification such that for all* $(\vec{y},\mathsf{p}) \in \mathrm{dom}(\Delta)$, $\Delta(\vec{y},\mathsf{p})$ *is in normal form. If* $\Delta \xrightarrow{\tau} \Delta'$, *then for all* $r \in \mathcal{R}_{\vec{y}}(\Delta')$ *either:*

- $r \in \mathcal{R}_{\vec{y}}(\Delta)$, *or*
- $\langle \mathsf{p}, \overline{\mathsf{y}}\mathsf{d}\rangle r \in \mathcal{R}_{\vec{y}}(\Delta)$, *or else*
- $\langle \mathsf{q}, \mathsf{y}\ \mathsf{d}\rangle r \in \mathcal{R}_{\vec{y}}(\Delta)$.

Proof. By induction on the structure of the proof $\Delta \xrightarrow{\tau} \Delta'$.

- [TCom$_1$]: $\Delta = \Delta_1, (\vec{y},\mathsf{p}) : \bigoplus_{i\in I} e_i \prec \overline{\mathsf{y}_i}\ \mathsf{d}_i.\mathbb{T}_i, \mathsf{y}_j : [\vec{\mathsf{d}}]$ and $\Delta' = \Delta_1, (\vec{y},\mathsf{p}) : \mathbb{T}_j, \mathsf{y}_j : [\vec{\mathsf{d}} \cdot \mathsf{d}_j]$. Then, by [RTCom$_1$], $r \in \mathcal{R}_{\vec{y}}(\Delta')$ implies $\langle \mathsf{p}, \overline{\mathsf{y}}\mathsf{d}\rangle r \in \mathcal{R}_{\vec{y}}(\Delta)$.
- [TCom$_2$]: $\Delta = \Delta_1, (\vec{y},\mathsf{p}) : \sum_{i\in I} e_i \prec \mathsf{y}_i\ \mathsf{d}_i.\mathbb{T}_i, \mathsf{y}_j : [\mathsf{d}_j \cdot \vec{\mathsf{d}}]$ and $\Delta' = \Delta_1, (\vec{y},\mathsf{p}) : \mathbb{T}_j, \mathsf{y}_j : [\vec{\mathsf{d}}]$. Then, by [RTCom$_2$], $r \in \mathcal{R}_{\vec{y}}(\Delta')$ implies $\langle \mathsf{p}, \mathsf{y}_j\mathsf{d}_j\rangle r \in \mathcal{R}_{\vec{y}}(\Delta)$.
- [TInit]: Then, $\Delta' = \Delta, (\vec{y'},\mathsf{p}_0) : \mathbb{T}_0, \ldots, (\vec{y'},\mathsf{p}_n) : \mathbb{T}_n, \vec{y'} : []$. By well-formedness conditions on specifications, $\vec{y} \subseteq \mathrm{dom}(\Delta)$ implies $\vec{y} \cap \vec{y'}$. The case follows by using Lemma B.3 to conclude that $\mathcal{R}_{\vec{y}}(\Delta') = \mathcal{R}_{\vec{y}}(\Delta)$.
- [TSeq]: Then, $\Delta = \Delta_1; \Delta_2$, and $\Delta' = \Delta_1'; \Delta_2$ with $\Delta_1 \xrightarrow{\tau} \Delta_1'$. By rule [RTSeq], $r = r_1 r_2$ with $r_1 \in \mathcal{R}_{\vec{y}}(\Delta_1')$ and $r_2 \in \mathcal{R}_{\vec{y}}(\Delta_2)$. By inductive hypothesis, $r_1 \in \mathcal{R}_{\vec{y}}(\Delta_1')$ implies $r_1 \in \mathcal{R}_{\vec{y}}(\Delta_1)$, $\langle \mathsf{p}, \overline{\mathsf{y}}\mathsf{d}\rangle r_1 \in \mathcal{R}_{\vec{y}}(\Delta_1)$, $\langle \mathsf{q}, \mathsf{y}\ \mathsf{d}\rangle r_1 \in \mathcal{R}_{\vec{y}}(\Delta_1)$, or $r_1 = r_1' r_2' r_3'$ and $r_1'[r_2']r_3' \in \mathcal{R}_{\vec{y}}(\Delta_1)$. Then, the proof is completed by using rule [RTSeq].
- [TLoop$_0$]: Then, $\Delta' = \Delta \upharpoonright \mathbb{U}$. By inspection of rules in Figure 14, we conclude that $r \in \mathcal{R}_{\vec{y}}(\Delta')$ implies $r = \epsilon$. Then, $r \in \mathcal{R}_{\vec{y}}(\Delta)$ by [RTEnd].
- [TLoop$_1$]: Then, $\Delta = \Delta_1^\star$ and $\Delta' = \Delta_1$. By [RTIt1], $r \in \mathcal{R}_{\vec{y}}(\Delta')$ implies $r \in \mathcal{R}_{\vec{y}}(\Delta)$.
- [TLoop$_2$]: Then, $\Delta = \Delta_1^\star$ and $\Delta' = \Delta_1; \Delta_1^\star$. By inspection of rules in Figure 14, we conclude that $r \in \mathcal{R}_{\vec{y}}(\Delta')$ implies $r = r_1 r_2$ with $r_1 \in \mathcal{R}_{\vec{y}}(\Delta_1)$ and $r_2 \in \mathcal{R}_{\vec{y}}(\Delta_1^\star)$. By [RTIt2], $r_1[r_2] \in \mathcal{R}_{\vec{y}}(\Delta)$. □

Lemma 8.6. *Let Δ a specification such that for all $(\vec{y},\mathsf{p}) \in \mathrm{dom}(\Delta)$, $\Delta(\vec{y},\mathsf{p})$ is in normal form. If $r \in \mathcal{R}_{\vec{y}}(\Delta)$ and $r \neq \epsilon$ then $\Delta \xrightarrow{\tau} \Delta'$ and either*

- *$r \in \mathcal{R}_{\vec{y}}(\Delta')$, or*
- *$r = \langle \mathsf{p}, \overline{\mathsf{y}}\mathsf{d}\rangle r'$ and $r' \in \mathcal{R}_{\vec{y}}(\Delta')$, or*
- *$r = \langle \mathsf{q}, \mathsf{y}\mathsf{d}\rangle r'$, or else and $r' \in \mathcal{R}_{\vec{y}}(\Delta')$.*

Proof. The proof follows by induction on the structure of proof $r \in \mathcal{R}_{\vec{y}}(\Delta)$.

- [RTCom$_1$] follows immediately by taking $r = \langle \mathsf{p}, \overline{\mathsf{y}}\mathsf{d}\rangle r'$ with $r' \in \mathcal{R}_{\vec{y}}(\Delta')$ and using [TCom$_1$].
- [RTCom$_2$] follows immediately by taking $r = \langle \mathsf{p}, \mathsf{y}\mathsf{d}\rangle r'$ with $r' \in \mathcal{R}_{\vec{y}}(\Delta')$ and using [TCom$_2$].
- [RTSeq] Then $\Delta = \Delta_1; \Delta_2$, $r = r_1 r_2$ with $r_1 \in \mathcal{R}_{\vec{y}}(\Delta_1)$ and $r_2 \in \mathcal{R}_{\vec{y}}(\Delta_2)$. The proof is completed by using inductive hypothesis on $r_1 \in \mathcal{R}_{\vec{y}}(\Delta_1)$ to conclude that $\Delta_1 \xrightarrow{\tau} \Delta_1'$. Then, by [TSeq], $\Delta = \Delta_1; \Delta_2 \xrightarrow{\tau} \Delta_1'; \Delta_2$. Finally, take $\Delta = \Delta_1'; \Delta_2$ and use rule [RTSeq] to conclude that either $r \in \mathcal{R}_{\vec{y}}(\Delta')$ or $r' \in \mathcal{R}_{\vec{y}}(\Delta')$.
- [RTIt1]. Then $\Delta = \Delta_1^\star$ and $r \in \mathcal{R}_{\vec{y}}(\Delta_1)$. The proof is completed by taking $\Delta' = \Delta_1$ and using rule [TLoop$_1$] to derive $\Delta \xrightarrow{\tau} \Delta'$.
- [RTIt2]. Then $\Delta = \Delta_1^\star$ and $r = r_1[r_2]$ with $r_1 \in \mathcal{R}_{\vec{y}}(\Delta_1)$ and $r_1 \in \mathcal{R}_{\vec{y}}(\Delta_1^\star)$. The proof is completed by taking $\Delta' = \Delta_1; \Delta_1^\star$ and using rule [TLoop$_2$] to derive $\Delta \xrightarrow{\tau} \Delta'$. □

B.2. **Runs of Specifications.**

Lemma B.5. *Let $r \in \mathcal{R}_{\vec{y}}(\Delta, (\vec{y},\mathsf{p}) : \sum_{i\in I} \mathsf{y}_i\ \mathsf{d}_i.\mathbb{T}_i)$ and $\mathsf{y} \in \vec{y}$ and $\mathsf{y} \neq \mathsf{y}_i$ for all $i \in I$, then $r \in \mathcal{R}_{\vec{y}}(\Delta, (\vec{y},\mathsf{p}) : \mathsf{y}\ \mathsf{d}; \mathbb{T} + \sum_{i\in I} \mathsf{y}_i\ \mathsf{d}_i.\mathbb{T}_i)$.*

Proof. By straightforward induction on the structure of the proof. □

Lemma B.6. *Let* $r \in \mathcal{R}_{\vec{y}}(\Delta, (\vec{y},\mathsf{p}) : \bigoplus_{i\in I} \overline{\mathsf{y}_i}\ \mathsf{d}_i.\mathbb{T}_i)$ *and* $\mathsf{y} \in \vec{y}$ *and* $\mathsf{y} \neq \mathsf{y}_i$ *for all* $i \in I$*, then* $r \in \mathcal{R}_{\vec{y}}(\Delta, (\vec{y},\mathsf{p}) : \mathsf{y}\ \mathsf{d}; \mathbb{T} \oplus \bigoplus_{i\in I} \overline{\mathsf{y}_i}\ \mathsf{d}_i.\mathbb{T}_i)$.

Proof. By straightforward induction on the structure of the proof. □

Lemma B.7. *Let* Δ *and* Δ' *be two specifications such that* $\partial\big(\Delta(\vec{y},\mathsf{p})\big) = \partial\big(\Delta(\vec{y},\mathsf{p})\big)$ *for all* $(\vec{y},\mathsf{p}) \in \mathrm{dom}(\Delta')$*. Then,* $\mathcal{R}_{\vec{y}}(\Delta) = \mathcal{R}_{\vec{y}}(\Delta')$.

Proof. It follows by straightforward induction on the derivation of $r \in \mathcal{R}_{\vec{y}}(\Delta)$ after noticing that guards are irrelevant for deriving runs. □

B.3. Specifications cover Global types.

Lemma B.8. *Let* $\mathcal{G}(\vec{y}) \triangleq \mathsf{G}$ *be a global type with* $\mathcal{P}(\mathsf{G}) = \{\mathsf{p}_0, \ldots, \mathsf{p}_n\}$ *and* $\Delta = (\vec{y},\mathsf{p}_0) : \mathsf{G}\upharpoonright \mathsf{p}_0, \ldots, (\vec{y},\mathsf{p}_n) : \mathsf{G}\upharpoonright \mathsf{p}_n, \vec{y} : []$*. If* $\mathcal{R}(\mathsf{G}) \subseteq \mathcal{R}_{\vec{y}}(\Delta)$*, then* $r \in \widetilde{\mathcal{R}}(\mathsf{G}^{*^f}) \subseteq \mathcal{R}_{\vec{y}}(\Delta^\star)$.

Proof. By induction on the structure of the derivation $r \in \widetilde{\mathcal{R}}(\mathsf{G}^{*^f})$

- [RG_1^*]: then $r \in \mathcal{R}(\mathsf{G})$. By hypothesis, $r \in \mathcal{R}_{\vec{y}}(\Delta)$. By [RTIt1], $r \in \mathcal{R}_{\vec{y}}(\Delta^\star)$.
- [RG_1^*]: then $r = r_1[r_2]$, $r_1 \in \mathcal{R}(\mathsf{G})$ and $r_2 \in \widetilde{\mathcal{R}}(\mathsf{G}^{*^f})$. By hypothesis, $r_1 \in \mathcal{R}_{\vec{y}}(\Delta)$. By inductive hypothesis $r_2 \in \mathcal{R}_{\vec{y}}(\Delta^\star)$. By [RTIt2], $r_1[r_2] \in \mathcal{R}_{\vec{y}}(\Delta^\star)$. □

Theorem 8.7 (Coverage & projections)**.** *Let* $\mathcal{G}(\vec{y}) \triangleq \mathsf{G}$ *be a global type with* $\mathcal{P}(\mathsf{G}) = \{\mathsf{p}_0, \ldots, \mathsf{p}_n\}$ *and* $\Delta = (\vec{y},\mathsf{p}_0) : \mathbb{T}_0, \ldots, (\vec{y},\mathsf{p}_n) : \mathbb{T}_n, \vec{y} : []$ *such that* $\partial(\mathbb{T}_i) = \mathsf{nf}(\mathsf{G}\upharpoonright \mathsf{p}_i)$ *for all* $0 \leq i \leq n$*. Then* $\mathcal{R}(\mathsf{G}) \Subset \mathcal{R}_{\vec{y}}(\Delta)$.

Proof. We show a stronger result proving that $\mathcal{R}(\mathsf{G}) \subseteq \mathcal{R}_{\vec{y}}(\Delta)$, which obviously implies $\mathcal{R}(\mathsf{G}) \Subset \mathcal{R}_{\vec{y}}(\Delta)$. We proceed by induction on the derivation of $r \in \mathcal{R}(\mathsf{G})$. We proceed by case analysis on the last applied rule in the derivation of $r \in \mathcal{R}(\mathsf{G})$.

- Case [RGEnd]: Follows straightforwardly from rule [RTEnd] observing that $\mathsf{G} = \mathsf{end}$ and Δ is end-only by the definition of projection (*cf.* on page 9).
- Case [RGCom]: Then, $\mathsf{G} = \sum_{i\in I} \mathsf{p} \twoheadrightarrow \mathsf{q}_i : \mathsf{y}_i\ \mathsf{d}_i; \mathsf{G}_i$ and $r = \langle \mathsf{p}, \overline{\mathsf{y}_h}\mathsf{d}_h, \sigma\rangle\langle \mathsf{q}_h, \mathsf{y}_h\mathsf{d}_h\rangle r'$ for an $h \in I$ and an $r' \in \mathcal{R}(\mathsf{G}_h)$.
  Then, letting $J_i = \{j \in I \mid \mathsf{q}_j = \mathsf{q}_i\}$ and $K_i = I \setminus J_i$ for all $i \in I$

$$\mathsf{G}\upharpoonright \mathsf{p} = \bigoplus_{i\in I} \overline{\mathsf{y}_i}\ \mathsf{d}_i.\mathsf{G}_i\upharpoonright \mathsf{p} \qquad\qquad \mathsf{G}\upharpoonright \mathsf{q}_i = \sum_{j\in J_i} \mathsf{y}_j\ \mathsf{d}_j.\mathsf{G}_j\upharpoonright \mathsf{r} + \sum_{k\in K_i} \mathsf{G}_k\upharpoonright \mathsf{r}$$

  By inductive hypothesis, we know that

$$r' \in \mathcal{R}_{\mathsf{y}}((\vec{y},\mathsf{p}_0) : \mathsf{G}_h\upharpoonright \mathsf{p}_0, \ldots, (\vec{y},\mathsf{p}_n) : \mathsf{G}_h\upharpoonright \mathsf{p}_n, \vec{y} : []) \tag{B.1}$$

  By repeatedly applying Lemma B.5 over (B.1), for each $\mathsf{p}_i \neq \mathsf{p}_h$, we conclude that

$$r' \in \mathcal{R}_{\mathsf{y}}(\Delta') \qquad \text{where} \qquad \Delta' : \begin{cases} \vec{y} \mapsto [] & \\ (\vec{y},\mathsf{p}) \mapsto \mathsf{G}_h\upharpoonright \mathsf{p} & \\ (\vec{y},\mathsf{q}_h) \mapsto \mathsf{G}_h\upharpoonright \mathsf{q}_h & \\ (\vec{y},\mathsf{q}) \mapsto \mathsf{G}\upharpoonright \mathsf{q} & \text{if } \mathsf{q} \notin \{\mathsf{p}, \mathsf{q}_h\} \\ \text{undefined} & \text{otherwise} \end{cases}$$

The proof is completed by using rules [RTCom$_1$] and [RTCom$_2$] as follows:

$$\dfrac{\dfrac{\overline{r' \in \mathcal{R}_{\mathsf{y}}(\Delta')}}{\langle \mathsf{q}_h, \mathsf{y}_h \mathsf{d}_h\rangle r' \ \in \mathcal{R}_{\mathsf{y}}(\Delta'[\vec{\mathsf{y}},\ (\vec{\mathsf{y}},\mathsf{q_h}) \mapsto [\mathsf{d}_h],\ \mathsf{G}\!\upharpoonright\!\mathsf{q}_h])}\,[\mathsf{RTCom}_2]}{\langle \mathsf{p}, \overline{\mathsf{y}_h}\mathsf{d}_h, \sigma\rangle\langle \mathsf{q}_h, \mathsf{y}_h \mathsf{d}_h\rangle r' \in \mathcal{R}_{\mathsf{y}}(\Delta)}\,[\mathsf{RTCom}_1]$$

- Case [RGSeq]: Then $\mathsf{G} = \mathsf{G}_1; \mathsf{G}_2$ and $r = r_1 r_2$ with $r_i \in \mathcal{R}(\mathsf{G}_i)$ for $i \in \{1,2\}$ and
$$r_i \in \mathcal{R}_{\mathsf{y}}((\vec{\mathsf{y}},\mathsf{p_0}) : \mathsf{G}_i\!\upharpoonright\!\mathsf{p}_0, \ldots, (\vec{\mathsf{y}},\mathsf{p_n}) : \mathsf{G}_i\!\upharpoonright\!\mathsf{p}_n, \vec{\mathsf{y}} : [])$$
by inductive hypothesis; hence, by rule [RTSeq]
$$r_1 r_2 \in \mathcal{R}_{\mathsf{y}}((\vec{\mathsf{y}},\mathsf{p_0}) : \mathsf{G}\!\upharpoonright\!\mathsf{p}_0, \ldots, (\vec{\mathsf{y}},\mathsf{p_n}) : \mathsf{G}\!\upharpoonright\!\mathsf{p}_n, \vec{\mathsf{y}} : [])$$
- Case [RGIter]: Then, $\mathsf{G} = \mathsf{G}_1^{*^f}$ with $\mathcal{P}(\mathsf{G}_1) = \{\mathsf{p}_0, \mathsf{p}_1, \ldots, \mathsf{p}_n\}$, $\mathtt{rdy}(\mathsf{G}_1) = \mathsf{p}_0$, $f(\mathsf{p}_i) = \mathsf{y}_i\ \mathsf{d}_i$ for all $1 \le i \le n$, and $r = r'\langle \mathsf{p}_0, \overline{\mathsf{y}_1}\mathsf{d}_1\rangle\langle \mathsf{p}_1, \mathsf{y}_1\mathsf{d}_1\rangle \ldots \langle \mathsf{p}_0, \overline{\mathsf{y}_n}\mathsf{d}_n\rangle\langle \mathsf{p}_n, \mathsf{y}_n\mathsf{d}_n\rangle$ with $r' \in \widetilde{\mathcal{R}}(\mathsf{G}^{*^f})$. Therefore,
$$\Delta = (\vec{\mathsf{y}},\mathsf{p_0}) : (\mathsf{G}_1\!\upharpoonright\!\mathsf{p}_0)^\star; \overline{\mathsf{y}_1}\ \mathsf{d}_1; \ldots; \overline{\mathsf{y}_n}\ \mathsf{d}_n, (\vec{\mathsf{y}},\mathsf{p_1}) : (\mathsf{G}_1\!\upharpoonright\!\mathsf{p}_1)^\star; \mathsf{y}_1\ \mathsf{d}_1, \ldots, (\vec{\mathsf{y}},\mathsf{p_n}) : (\mathsf{G}_1\!\upharpoonright\!\mathsf{p}_1)^\star; \mathsf{y}_n\ \mathsf{d}_n, \vec{\mathsf{y}} : []$$
Note that Δ can be written as the sequential composition $\Delta = \Delta_1^\star; \Delta_2$ where
$$\begin{aligned}\Delta_1 = \ & (\vec{\mathsf{y}},\mathsf{p_0}) : (\mathsf{G}_1\!\upharpoonright\!\mathsf{p}_0), (\vec{\mathsf{y}},\mathsf{p_1}) : (\mathsf{G}_1\!\upharpoonright\!\mathsf{p}_1), \ldots, (\vec{\mathsf{y}},\mathsf{p_n}) : (\mathsf{G}_1\!\upharpoonright\!\mathsf{p}_n), \vec{\mathsf{y}} : [] \\ \Delta_2 = \ & (\vec{\mathsf{y}},\mathsf{p_0}) : \overline{\mathsf{y}_1}\ \mathsf{d}_1; \ldots; \overline{\mathsf{y}_n}\ \mathsf{d}_n, (\vec{\mathsf{y}},\mathsf{p_1}) : \mathsf{y}_1\ \mathsf{d}_1, \ldots, (\vec{\mathsf{y}},\mathsf{p_n}) : \mathsf{y}_n\ \mathsf{d}_n, \vec{\mathsf{y}} : []\end{aligned}$$
Then, by repeated used of rules [RTCom$_1$] and [RTCom$_2$] we can build a proof for $r'' = \langle \mathsf{p}_0, \overline{\mathsf{y}_1}\mathsf{d}_1\rangle\langle \mathsf{p}_1, \mathsf{y}_1\mathsf{d}_1\rangle \ldots \langle \mathsf{p}_0, \overline{\mathsf{y}_n}\mathsf{d}_n\rangle\langle \mathsf{p}_n, \mathsf{y}_n\mathsf{d}_n\rangle \in \mathcal{R}_{\vec{\mathsf{y}}}(\Delta_2)$, as illustrated by the following sketch

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\epsilon \in \mathcal{R}_{\vec{\mathsf{y}}}((\vec{\mathsf{y}},\mathsf{p_0}) : \mathsf{end}, \ldots, (\vec{\mathsf{y}},\mathsf{p_n}) : \mathsf{end}, \vec{\mathsf{y}} : [])}\,[\mathsf{RTEnd}]}{\vdots}}{\langle \mathsf{p}_0, \overline{\mathsf{y}_2}\mathsf{d}_2\rangle\langle \mathsf{p}_2, \mathsf{y}_2\mathsf{d}_2\rangle \ldots \langle \mathsf{p}_0, \overline{\mathsf{y}_n}\mathsf{d}_n\rangle\langle \mathsf{p}_n, \mathsf{y}_n\mathsf{d}_n\rangle \in \mathcal{R}_{\vec{\mathsf{y}}}(\Delta_2[(\vec{\mathsf{y}},\mathsf{p_1}),\ (\vec{\mathsf{y}},\mathsf{p_0}) \mapsto \mathsf{end},\ \overline{\mathsf{y}_2}\ \mathsf{d}_2; \ldots; \overline{\mathsf{y}_n}\ \mathsf{d}_n])}\,[\mathsf{RTCom}_1]}{\langle \mathsf{p}_1, \mathsf{y}_1\mathsf{d}_1\rangle\langle \mathsf{p}_0, \overline{\mathsf{y}_2}\mathsf{d}_2\rangle\langle \mathsf{p}_2, \mathsf{y}_2\mathsf{d}_2\rangle \ldots \langle \mathsf{p}_0, \overline{\mathsf{y}_n}\mathsf{d}_n\rangle\langle \mathsf{p}_n, \mathsf{y}_n\mathsf{d}_n\rangle \in \mathcal{R}_{\vec{\mathsf{y}}}(\Delta_2[\mathsf{y}_1,\ (\vec{\mathsf{y}},\mathsf{p_0}) \mapsto [\mathsf{d}_1 @ \mathsf{p}_0],\ \overline{\mathsf{y}_2}\ \mathsf{d}_2; \ldots; \overline{\mathsf{y}_n}\ \mathsf{d}_n])}\,[\mathsf{RTCom}_2]}{r'' \in \mathcal{R}_{\vec{\mathsf{y}}}(\Delta_2)}\,[\mathsf{RTCom}_1]$$

We now show that $r' \in \widetilde{\mathcal{R}}(\mathsf{G}_1^{*^f})$ implies $r' \in \mathcal{R}_{\vec{\mathsf{y}}}(\Delta_1^\star)$. By inductive hypothesis, $\mathcal{R}(\mathsf{G}_1) \subseteq \mathcal{R}_{\vec{\mathsf{y}}}(\Delta_1)$. By Lemma B.8, $\widetilde{\mathcal{R}}(\mathsf{G}^{*^f}) \subseteq \mathcal{R}_{\vec{\mathsf{y}}}(\Delta_1^\star)$. Therefore, $r' \in \widetilde{\mathcal{R}}(\mathsf{G}^{*^f})$ implies $r' \in \mathcal{R}_{\vec{\mathsf{y}}}(\Delta_1^\star)$. By [RTSeq], $r = r'r'' \in \mathcal{R}_{\vec{\mathsf{y}}}(\Delta_1^\star; \Delta_2)$ since $r' \in \mathcal{R}_{\vec{\mathsf{y}}}(\Delta_1^\star)$ and $r'' \in \mathcal{R}_{\vec{\mathsf{y}}}(\Delta_2)$. □

B.4. **Implementations cover specifications.**

Definition B.9 (Viable types)**.** A pseudo-type $\mathbb{T}$ is *viable* if either of the following holds

- $\mathsf{nf}(\mathbb{T}) = e \prec \mathsf{end}$
- $\mathsf{nf}(\mathbb{T}) = \bigoplus_{i\in I} e_i \prec \overline{\mathsf{y}_i}\ \mathsf{d}_i.\mathbb{T}_i$ or $\mathsf{nf}(\mathbb{T}) = \sum_{i\in I} e_i \prec \mathsf{y}_i\ \mathsf{d}_i.\mathbb{T}_i$ with $\mathbb{T}_i$ viable for all $i \in I$
- $\mathsf{nf}(\mathbb{T}) = \mathbb{T}_1^\star; \mathbb{T}_2$ with $\mathbb{T}_1$ and $\mathbb{T}_2$ viable, and either $\mathbb{T}_1$ and $\mathbb{T}_2$ passively compatible or $\mathbb{T}_1 = \bigoplus_{i\in I} e_i \prec \overline{\mathsf{y}_i}\ \mathsf{d}_i.\mathbb{T}_i$.

A specification Δ is *viable* when every type in Δ is viable.

Lemma B.10. *Let $\mathcal{G}(\vec{\mathsf{y}})$ a global type and Δ a specification such that (i)* $\mathrm{dom}(\Delta) = \{(\vec{\mathsf{y}},\mathsf{q}) \mid \mathsf{q} \in \mathcal{P}(\mathcal{G})\}$ *and (ii) $\Delta : (\vec{\mathsf{y}},\mathsf{q}) \mapsto \mathcal{G}\!\upharpoonright\!\mathsf{q}$ for all $\mathsf{q} \in \mathcal{P}(\mathcal{G})$. Then Δ is viable.*

In what follows we write $\langle S,\sigma\rangle \xrightarrow{E\vdash\widetilde{\alpha}} \langle S',\sigma'\rangle$ with $E = e_1 \wedge \ldots \wedge e_n$ and $\widetilde{\alpha} = \alpha_1 \ldots \alpha_n$ for the sequence $\langle S,\sigma\rangle \xrightarrow{e_1\vdash\alpha_1} \langle S_2,\sigma_2\rangle \xrightarrow{e_2\vdash\alpha_2} \ldots \xrightarrow{e_n\vdash\alpha_n} \langle S',\sigma\rangle$. Let $\mathtt{n}(\alpha) = \mathtt{fy}(\alpha) \cup \mathtt{bn}(\alpha)$; the definitions of $\mathtt{n}(\_)$, $\mathtt{fy}(\_)$ and $\mathtt{bn}(\_)$ straightforwardly extend to sequences of labels.

Lemma B.11. *Let $\mathbb{T}$ be a viable pseudo-type and e such that $\mathsf{nf}(e,\mathbb{T}) = \mathbb{T}$. Then, there exist Γ and P and Δ and $\mathbb{T}'$ such that for any $\vec{\mathsf{y}} \supseteq \mathtt{fy}(\mathbb{T})$ and p it holds that $e \lrcorner \Gamma \vdash P \triangleright \Delta, (\vec{\mathsf{y}},\mathsf{p}) : \mathbb{T}'$ and $\partial(\mathbb{T}) = \partial(\mathbb{T}')$.*

Proof. The proof follows by induction on the structure of $\mathbb{T}$.

- $\mathbb{T} = e_j \prec \mathsf{end}$. It follows immediately by using rule [VEnd].
- $\mathbb{T} = \bigoplus_{i\in I} e_i \prec \overline{\mathsf{y}_i}\ \mathsf{d}_i.\mathbb{T}_i$. Note that $\mathsf{nf}(e,\mathbb{T}) = \mathbb{T}$ implies $\mathsf{nf}(e,\mathbb{T}_i) = \mathbb{T}_i$ for all $i \in I$. Then, by inductive hypothesis there exist Γ_i, P_i, Δ_i, $\mathbb{T}_i$ for any $\vec{\mathsf{y}} \supseteq \mathtt{fy}(\mathbb{T}_i)$ and p it holds that $e \lrcorner \Gamma_i \vdash P_i \triangleright \Delta_i, (\vec{\mathsf{y}},\mathsf{p}) : \mathbb{T}_i'$ and $\partial(\mathbb{T}_i) = \partial(\mathbb{T}_i')$. Then, take s and x fresh, i.e., $s \notin \mathrm{dom}(\Delta_0',\ldots,\Delta_n')$ and $x \notin \mathrm{dom}(\Gamma_0',\ldots,\Gamma_n')$ and note that
$$e \lrcorner \Gamma_i, x : \mathsf{d} \vdash P_i \triangleright \Delta_i, (\vec{\mathsf{y}},\mathsf{p}) : \mathbb{T}_i', (s,\mathsf{p}') : e \prec \mathsf{end}$$
for any d and p'. By Lemma A.14,
$$e \wedge e_i' \lrcorner \Gamma_i, x : \mathsf{d} \vdash P_i \triangleright \Delta_i', (\vec{\mathsf{y}},\mathsf{p}) : \mathbb{T}_i', (s,\mathsf{p}') : e' \prec \mathsf{end}$$
with $e_i' = x \neq 0 \wedge \ldots \wedge x \neq i-1 \wedge x = i$ and
$$\begin{array}{lcl} \Delta_i'(\vec{\mathsf{y}},\mathsf{p}) & = & e \wedge e_i' \prec \mathsf{end}; \Delta_i(\vec{\mathsf{y}},\mathsf{p}) \\ \mathbb{T}_i' & = & e \wedge e_i' \prec \mathsf{end}; \mathbb{T}_i \\ e' \prec \mathsf{end} & = & e \wedge e_i' \prec \mathsf{end}; e \prec \mathsf{end} \end{array}$$
By definition of $\mathsf{nf}(\_,\_)$, $\mathbb{T}_i' = \mathsf{nf}(e \wedge e_i', \mathbb{T}_i)$. Since $x \notin \mathtt{fn}(\mathbb{T})$ and $\mathsf{nf}(e,\mathbb{T}) = \mathbb{T}$ we have that $\partial(\mathbb{T}_i') = \partial(\mathbb{T}_i)$. Moreover, by typing rules [VSend] and [VSeq],
$$e \wedge e_i' \lrcorner \Gamma_i, x : \mathsf{d} \vdash \overline{\mathsf{y}_i}\, c_i; P_i \triangleright \Delta_i''$$
such that c_i is some constant of type d_i and
$$\Delta_i''(\vec{\mathsf{y}},\mathsf{p}) = e \wedge e_i' \prec \overline{\mathsf{y}_i}\ \mathsf{d}_i; \Delta_i'(\vec{\mathsf{y}},\mathsf{p})$$
Then, take $\mathbb{T}' = \bigoplus_{i\in I} \Delta_i''(\vec{\mathsf{y}},\mathsf{p})$. Note that $\mathbb{T}'$ is well-defined (because the guards in all types are different) and $\partial(\mathbb{T}') = \partial(\mathbb{T})$ because $\partial(\mathbb{T}_i') = \partial(\mathbb{T}_i)$ holds for every i.

Finally, define $\Delta = \Delta_0, \ldots, \Delta_n, (\vec{y},\mathsf{p}) : \mathbb{T}', (s,\mathsf{p}') : e \prec \mathsf{end}$, and $\Delta = \Delta_0, \ldots, \Delta_n$ and $P = s(x).\mathsf{if}\ x = 0\ \mathsf{then}\ P_0\ \mathsf{else\ if}\ x = 1\ \mathsf{then}\ P_1\ \mathsf{else}\ \ldots$. The proof is concluded by straightforward use of typing rules.

- $\sum_{i\in I} e_i \prec \mathsf{y}_i\ \mathsf{d}_i.\mathbb{T}_i$. It follows analogously to the previous case.
- $\mathbb{T}_1^\star;\mathbb{T}_2$ with $\mathbb{T}_1$ with $\mathbb{T}_1 = \bigoplus_{i\in I} e_i \prec \overline{\mathsf{y}_i}\ \mathsf{d}_i.\mathbb{T}_i$. It follows by inductive hypothesis on both $\mathbb{T}_1$ and $\mathbb{T}_2$ to conclude that there exist P_1 and P_2 and by taking $P = \mathsf{for}\ x\ \mathsf{in}\ \ell\ \mathsf{do}\ P_1; P_2$.
- $\mathbb{T}_1^\star;\mathbb{T}_2$ and $\mathbb{T}_1$ and $\mathbb{T}_2$ passively compatible. It follows by inductive hypothesis on both $\mathbb{T}_1$ and $\mathbb{T}_2$ to conclude that there exist P_1 and P_2 and by taking $P = \mathsf{repeat}\ P_1\ \mathsf{until}\ P_2$.

□

Lemma B.12. *If $e \llcorner \Gamma \vdash P \triangleright \Delta$ and $x \notin \mathtt{fx}(P)$ then, for all $x' \in \mathbb{X}$, $e \llcorner \Gamma|_{-x'}[x \mapsto \Gamma(x')] \vdash P\{x/x'\} \triangleright \Delta'$ where $\Delta'|_{\mathbb{U}\cup\mathbb{Y}} = \Delta|_{\mathbb{U}\cup\mathbb{Y}}$ and for all $(\vec{y},\mathsf{p}) \in \mathrm{dom}(\Delta)$, $\Delta'(\vec{y},\mathsf{p}) = \Delta(\vec{y},\mathsf{p})\{x/x'\}$.*

Proof. Trivial, observing that typing does not depend on the identity of the variables used in processes. □

Definition B.13. Let $\iota_{\vec{y}@u}$ be a ι-implementation of a global type $\mathcal{G}$ at u, e a guard, Γ an environemnt, and Δ a specification. We say that $\iota_{\vec{y}@u}$, $\Gamma' \supseteq \Gamma$, and specification $\Delta' = (\Delta_\mathsf{q})_{\mathsf{q}\in\mathcal{P}(\mathcal{G})}$ are *compatible with e, Γ, and Δ* if

- the queue on each $\mathsf{y} \in \vec{y}$ in $\iota_{\vec{y}@u}$ is typed as $\Delta(\mathsf{y})$ and
- for all $\mathsf{q} \in \mathcal{P}(\mathcal{G})$

$$e \llcorner \Gamma' \vdash \iota(\mathsf{q}) \triangleright \Delta_\mathsf{q} \text{ and } \partial\big(\Delta_\mathsf{q}(\vec{y},\mathsf{q})\big) = \partial\big(\Delta(\vec{y},\mathsf{q})\big) \tag{B.2}$$

Lemma B.14. *Let $\mathcal{G}(\vec{y})$ be a global type, Δ be a viable specification such that $\vec{y} \subseteq \mathrm{dom}(\Delta)$ and $(\vec{y},\mathsf{q}) \in \mathrm{dom}(\Delta)$ for all $\mathsf{q} \in \mathcal{P}(\mathcal{G})$, and let $e \llcorner \Gamma \vdash P \triangleright \Delta'$ be a judgement where $\Delta'(\vec{y},\mathsf{p}) = \Delta(\vec{y},\mathsf{p})$ for a participant $\mathsf{p} \in \mathcal{P}(\mathcal{G})$. For all $r \in \mathcal{R}_{\vec{y}}(\Delta)$ there are $\iota_{\vec{y}@u}$, an environment Γ', and a specification Δ' compatible with e, Γ, and Δ such that*

(1) $\iota(\mathsf{p}) = P$
(2) *for all $\sigma \ltimes (\Delta'; e; \iota_{\vec{y}@u}; \Gamma')$ there is $r' \in \mathcal{R}\langle\iota_{\vec{y}@u}, \sigma\rangle$ such that $r \lessdot r'$.*

Proof. By induction on the structure of the derivation $r \in \mathcal{R}_{\vec{y}}(\Delta)$. In the proof, $\vdash \iota(\vec{y}) \colon \Delta(\vec{y})$ shortens $\vdash \mathsf{y}[\vec{v}] \colon \Delta(\mathsf{y})$ for all $\mathsf{y} \in \vec{y}$ where $\mathsf{y}[\vec{v}]$ is the queue on y in $\iota_{\vec{y}@u}$.

- [RTEnd] Then, $r = \epsilon$ and for all $\mathsf{q} \in \mathcal{P}(\mathcal{G})$, $\Delta(\vec{y},\mathsf{q}) = e_\mathsf{q} \prec \mathsf{end}$ and $\Delta(\vec{y}) = []$. Now we show that each condition holds.
  (1) Take $\iota(\mathsf{p}) = P$, and $\iota(\mathsf{q}) = 0$ for all $\mathsf{q} \neq \mathsf{p}$, and $\vdash \iota(\vec{y}) \colon []$.
  (2) Let $\Gamma' = \Gamma$, $\Delta_\mathsf{p} = \Delta'$ and $\Delta_\mathsf{q} : (\vec{y},\mathsf{q}) \mapsto e \prec \mathsf{end}$ for all $\mathsf{q} \neq \mathsf{p}$.
    (a) Then condition (B.2) holds for participant p by the hypothesis $e \llcorner \Gamma \vdash P \triangleright \Delta' = \Delta_\mathsf{p}$, and straightforwardly for participants $\mathsf{q} \neq \mathsf{p}$ observing that $e \llcorner \Gamma \vdash \iota(\mathsf{q}) \triangleright \Delta_\mathsf{q}$ and guard removals coincide since all local types are end.
    (b) For condition (2) it is enough to take any store σ mapping each free variable x of P in a value of type $\Gamma(x)$ so that e evaluates to $\mathtt{true}$ in σ (the existence of such values is guaranteed by the typing of P). This also entails (4) of Definition 8.1. Finally, note that $r' = \epsilon \in \mathcal{R}\langle\iota_{\vec{y}@u}, \sigma\rangle$ from [REnd] and $r \lessdot r'$ follows from [$\lessdot$-emp].
- [RTCom$_1$] There is $\mathsf{q} \in \mathcal{P}(\mathcal{G})$ such that $\Delta(\vec{y},\mathsf{q}) = \bigoplus_{i\in I} e_i \prec \overline{\mathsf{y}_i}\ \mathsf{d}_i.\mathbb{T}_i$ and $r = \langle \mathsf{q}, \overline{\mathsf{y}_j}\mathsf{d}_j\rangle r_1$ with $j \in I$ and $r_1 \in \mathcal{R}_{\vec{y}}(\Delta'')$ where $\Delta'' = \Delta[(\mathsf{y},\mathsf{q}),\ \vec{y} \mapsto \mathbb{T}_j,\ [\vec{\mathsf{d}} \cdot \mathsf{d}_j]]$. Note that Δ'' is viable and that $\Delta(\vec{y},\mathsf{q}) \neq \mathtt{false} \prec \mathsf{end}$, which implies $\neg(e_j \iff \mathtt{false})$.

We distinguish two cases.

Case $\mathsf{q} = \mathsf{p}$. By Lemma A.21, we know that for any σ_0 such that $\sigma_0 \ltimes (e; \Gamma; P; \Delta')$ it holds that $\langle P, \sigma_0\rangle \xrightarrow{E \vdash \widetilde{\beta}} \langle P'', \sigma_1\rangle \xrightarrow{e' \vdash \overline{\mathsf{y}_j}\mathsf{v}} \langle P', \sigma_2\rangle$ with $\vdash \mathsf{v}\colon \mathsf{d}_j$, and $\mathtt{n}(\widetilde{\beta}) \cap \vec{\mathsf{y}} = \emptyset$, and $\neg(e \wedge E \wedge e' \iff \mathtt{false})$. By applying Theorem 8.2 to the hypothesis $e \lrcorner \Gamma \vdash P \triangleright \Delta'$ and σ_0, we have

$$\Delta' \xhookrightarrow{\widetilde{\beta}'} \Delta'_1 \xrightarrow{\overline{\mathsf{y}_j}\mathsf{d}_j} \Delta'_2, \quad e \wedge E \wedge e' \lrcorner \Gamma'' \vdash P' \triangleright \Delta'_2, \quad \text{and} \quad \Gamma \subseteq \Gamma'' \tag{B.3}$$

(where the sequence $\widetilde{\beta}'$ is the sequence $\widetilde{\beta}$ where expressions are replaced by their sorts). By Lemma B.4, $\mathtt{n}(\widetilde{\beta}') \cap \vec{\mathsf{y}} = \emptyset$ implies $\Delta'_1(\vec{\mathsf{y}},\mathsf{r}) = \Delta'(\vec{\mathsf{y}},\mathsf{r}) = \Delta(\vec{\mathsf{y}},\mathsf{r})$ for all $\mathsf{r} \in \mathcal{P}(\mathcal{G})$. Moreover, $\Delta'_2(\vec{\mathsf{y}},\mathsf{p}) = \mathbb{T}_j$ because of the reduction $\Delta'_1 \xrightarrow{\overline{\mathsf{y}_j}\mathsf{d}_j} \Delta'_2$. By inductive hypothesis on $r_1 \in \mathcal{R}_{\vec{\mathsf{y}}}(\Delta'')$ and (B.3), we know that there are $\iota^{\mathtt{ih}}{}_{\vec{\mathsf{y}}@u}$, $\Gamma^{\mathtt{ih}} \supseteq \Gamma''$, and $\Delta^{\mathtt{ih}} = (\Delta^{\mathtt{ih}}_{\mathsf{r}})_{\mathsf{r}\in\mathcal{P}(\mathcal{G})}$ compatible with $e \wedge E \wedge e'$, Γ'', and Δ'_2. Namely,
(1) $\iota^{\mathtt{ih}}(\mathsf{p}) = P'$ and
(2) $e \wedge E \wedge e' \lrcorner \Gamma^{\mathtt{ih}} \vdash \iota^{\mathtt{ih}}(\mathsf{r}) \triangleright \Delta^{\mathtt{ih}}_{\mathsf{r}}$ and $\partial\big(\Delta^{\mathtt{ih}}_{\mathsf{r}}(\vec{\mathsf{y}},\mathsf{r})\big) = \partial\big(\Delta''(\vec{\mathsf{y}},\mathsf{r})\big)$, for all $\mathsf{r} \in \mathcal{P}(\mathcal{G})$
(3) for all $\sigma^{\mathtt{ih}} \ltimes (\Delta^{\mathtt{ih}}; e \wedge E \wedge e'; \iota^{\mathtt{ih}}{}_{\vec{\mathsf{y}}@u}; \Gamma^{\mathtt{ih}})$ there is $r^{\mathtt{ih}}_1 \in \mathcal{R}\langle \iota^{\mathtt{ih}}{}_{\vec{\mathsf{y}}@u}, \sigma^{\mathtt{ih}}\rangle$ such that $r_1 \lessdot r^{\mathtt{ih}}_1$.
Then, let

- $\iota(\mathsf{p}) = P$, and $\iota(\mathsf{q}) = \iota^{\mathtt{ih}}(\mathsf{q})$ for all $\mathsf{q} \neq \mathsf{p} \in \mathcal{P}(\mathcal{G})$, and $\iota(\mathsf{y}_j) = [\vec{\mathsf{v}}]$ where $\iota^{\mathtt{ih}}(\mathsf{y}_j) = [\vec{\mathsf{v}} \cdot \mathsf{v}]$ (note that (1) and the definition of Δ'' imply that $\iota^{\mathtt{ih}}(\mathsf{y}_j)$ is a non-empty queue); this implies $\vdash \iota^{\mathtt{ih}}(\mathsf{y}_j)\colon [\vec{\mathsf{d}} \cdot \mathsf{d}]$ since $[\vec{\mathsf{d}} \cdot \mathsf{d}] = \Delta''(\mathsf{y}_j)$. Observe that for all $\mathsf{y} \in \vec{\mathsf{y}} \setminus \{\mathsf{y}_j\}$ we have $\vdash \iota(\mathsf{y})\colon \Delta''(\mathsf{y})$ since $\iota^{\mathtt{ih}}(\mathsf{y}) = \iota(\mathsf{y})$ and $\Delta(\mathsf{y}) = \Delta''(\mathsf{y})$.
- Note that $\Gamma^{\mathtt{ih}} \supseteq \Gamma$ since $\Gamma^{\mathtt{ih}} \supseteq \Gamma''$ and $\Gamma'' \supseteq \Gamma$ by (B.3). Let $\Delta_{\mathsf{p}} = \Delta'$ and $\Delta_{\mathsf{r}} = \Delta'_{\mathsf{r}}$ for all $\mathsf{r} \neq \mathsf{p}$; then
  (1) by the validity of $e \lrcorner \Gamma \vdash P \triangleright \Delta'$ and the fact that $\Gamma^{\mathtt{ih}} \supseteq \Gamma$, we have $e \lrcorner \Gamma^{\mathtt{ih}} \vdash \iota(\mathsf{p}) \triangleright \Delta_{\mathsf{p}}$ applying Lemma A.10. For $\mathsf{r} \neq \mathsf{p} \in \mathcal{P}(\mathcal{G})$, the validity of the judgement in (2) implies the validity of $e \lrcorner \Gamma \vdash \iota(\mathsf{r}) \triangleright \Delta_{\mathsf{r}}$ because $\mathrm{var}(E \wedge e')$ are not bound in $\iota(\mathsf{r})$. Also, $\partial\big(\Delta_{\mathsf{r}}(\vec{\mathsf{y}},\mathsf{r})\big) = \partial\big(\Delta''(\vec{\mathsf{y}},\mathsf{r})\big) = \partial\big(\Delta(\vec{\mathsf{y}},\mathsf{r})\big)$ where the first equality holds by the inductive hypothesis and the second by (2).
  (2) Let σ be $\sigma^{\mathtt{ih}}$ restricted on $\bigcup_{\mathsf{r}\in\mathcal{P}(\mathcal{G})} \mathrm{dom}(\Delta_{\mathsf{r}}|_{\mathbb{Y}}) \cup \mathrm{dom}(\Gamma^{\mathtt{ih}})$. Then the first three conditions of Definition 8.1 are trivially satisfied, e evaluates to $\mathtt{true}$ in σ since $\sigma|_{\mathrm{var}(e)} = \sigma^{\mathtt{ih}}|_{\mathrm{var}(e)}$, and likewise for the last condition of consistency.
- Note that $r = \langle \mathsf{p}, \overline{\mathsf{y}_j}\mathsf{d}_j\rangle r_1 \lessdot \langle \mathsf{p}, \overline{\mathsf{y}_j}\mathsf{d}_j\rangle r^{\mathtt{ih}}_1 = r'$ follows from [$\lessdot$-cmp] and (3). The fact that $r' \in \iota_{\vec{\mathsf{y}}@u}$ follows by repeated applications of rule [RExt] (once for any $\beta \in \widetilde{\beta}$) followed by an application of [RSnd].

Case $\mathsf{q} \neq \mathsf{p}$. By inductive hypothesis on $r_1 \in \mathcal{R}_{\vec{\mathsf{y}}}(\Delta'')$ and (B.3), we know that there are $\iota^{\mathtt{ih}}{}_{\vec{\mathsf{y}}@u}$, $\Gamma^{\mathtt{ih}} \supseteq \Gamma''$, and $\Delta^{\mathtt{ih}} = (\Delta^{\mathtt{ih}}_{\mathsf{r}})_{\mathsf{r}\in\mathcal{P}(\mathcal{G})}$ compatible with $e \wedge E \wedge e'$, Γ'', and Δ'_2. Namely,
(1) $\iota^{\mathtt{ih}}(\mathsf{p}) = P$
(2) $e \lrcorner \Gamma^{\mathtt{ih}} \vdash \iota^{\mathtt{ih}}(\mathsf{r}) \triangleright \Delta^{\mathtt{ih}}_{\mathsf{r}}$ and $\partial\big(\Delta^{\mathtt{ih}}_{\mathsf{r}}(\vec{\mathsf{y}},\mathsf{r})\big) = \partial\big(\Delta''(\vec{\mathsf{y}},\mathsf{r})\big)$, for all $\mathsf{r} \in \mathcal{P}(\mathcal{G})$
(3) for all store $\sigma^{\mathtt{ih}} \ltimes (\bigcup_{\mathsf{r}\in\mathcal{P}(\mathcal{G})} \Delta^{\mathtt{ih}}_{\mathsf{r}}; e; \iota^{\mathtt{ih}}{}_{\vec{\mathsf{y}}@u}; \Gamma^{\mathtt{ih}})$ there is $r^{\mathtt{ih}}_1 \in \mathcal{R}\langle \iota^{\mathtt{ih}}{}_{\vec{\mathsf{y}}@u}, \sigma^{\mathtt{ih}}\rangle$ such that $r_1 \lessdot r^{\mathtt{ih}}_1$.
Then, let

– $\iota(\mathsf{y}_j) = [\vec{\mathsf{v}}]$ where $\iota^{\mathtt{ih}}(\mathsf{y}_j) = [\vec{\mathsf{v}} \cdot \mathsf{v}]$ (the proof that queues $\mathsf{y} \in \vec{\mathsf{y}}$ are typed by Δ'' is as in the previous case $\mathsf{p} = \mathsf{q}$) and

$$\iota : \begin{cases} \mathsf{p} \mapsto P \\ \mathsf{r} \mapsto \iota^{\mathtt{ih}}(\mathsf{r}) & \forall \mathsf{r} \in \mathcal{P}(\mathcal{G}) \setminus \{\mathsf{p}, \mathsf{q}\} \\ \mathsf{q} \mapsto s(x).\mathsf{if}\ x\ \mathsf{then}\ \overline{\mathsf{y}_j}\,\mathsf{v}; \iota^{\mathtt{ih}}(\mathsf{q})\ \mathsf{else}\ Q \end{cases}$$

with s and x fresh (formally, $s \notin \bigcup_{\mathsf{r} \in \mathcal{P}(\mathcal{G})} \mathrm{dom}(\Delta^{\mathtt{ih}}_{\mathsf{r}})$, and $x \notin \mathrm{dom}(\Gamma)$, and $\vdash \mathsf{v} \colon \mathsf{d}_j$) and Q a process such that the following judgement holds:

$$e \lrcorner \Gamma_Q \ \vdash\ Q \ \triangleright\ \Delta_Q \quad \text{where} \quad \Delta_Q(\vec{\mathsf{y}},\mathsf{q}) = \bigoplus_{i \in I \setminus \{j\}} e_i \prec \overline{\mathsf{y}_i}\ \mathsf{d}_i.\mathbb{T}_i$$

The above judgment exists for the type $\mathbb{T} = \bigoplus_{i \in I \setminus \{j\}} e_i \prec \overline{\mathsf{y}_i}\ \mathsf{d}_i.\mathbb{T}_i$ by Lemma B.11, since $\mathsf{nf}(e, \mathbb{T}) = \mathbb{T}$ applying Lemma A.15 to the hypothesis $e \lrcorner \Gamma \ \vdash\ P \triangleright \Delta'$. By Lemma B.12 wlog we can assume that $\mathtt{fx}(Q)$ are fresh.

– Let $\Gamma' = \Gamma \cup \Gamma_Q$ and

$$\begin{aligned} \Delta_{\mathsf{p}} &= \Delta' \\ \Delta_{\mathsf{q}} &= \Delta^{\mathtt{ih}}_{\mathsf{q}}[\Delta_Q][(\mathsf{y},\mathsf{q}) \mapsto \Delta(\mathsf{y},\mathsf{q})][(s,\_) \mapsto s\ \mathsf{bool}; \mathsf{end}] \\ \Delta_{\mathsf{r}} &= \Delta^{\mathtt{ih}}_{\mathsf{r}} \qquad \textit{if } \mathsf{r} \notin \{\mathsf{p}, \mathsf{q}\} \end{aligned}$$

(1) by the validity of $e \lrcorner \Gamma \ \vdash\ P \triangleright \Delta'$ and the fact that $\Gamma^{\mathtt{ih}} \supseteq \Gamma$, we have $e \lrcorner \Gamma^{\mathtt{ih}} \ \vdash\ \iota(\mathsf{p}) \triangleright \Delta_{\mathsf{p}}$ applying Lemma A.10. For all $(\mathsf{y},\mathsf{r}) \in \mathrm{dom}(\Delta')$ with $\mathsf{r} \in \mathcal{P}(\mathcal{G}) \setminus \{\mathsf{p}, \mathsf{q}\}$ we have that $\Delta''(\mathsf{y},\mathsf{r}) = \Delta'(\mathsf{y},\mathsf{r})$ and $\Im\big(\Delta_{\mathsf{r}}(\vec{\mathsf{y}},\mathsf{r})\big) = \Im\big(\Delta''(\vec{\mathsf{y}},\mathsf{r})\big) = \Im\big(\Delta(\vec{\mathsf{y}},\mathsf{r})\big)$ by the inductive hypothesis. The validity of $e \lrcorner \Gamma' \ \vdash\ \iota(\mathsf{q}) \ \triangleright\ \Delta_{\mathsf{q}}$ holds by weakening (Lemma B.1) and the application of rules [VIf] and [VRcv] to the judgment $e \lrcorner \Gamma_Q \ \vdash\ Q \ \triangleright\ \Delta_Q$ above.

(2) Let σ be an extension of $\sigma^{\mathtt{ih}}$ with assignments to the free variables of Q so that no guard in Q is falsified. Then the first three conditions of Definition 8.1 trivially follow from the inductive hypothesis, and the last condition follows from Lemma A.19.

– From (3) we know that $r_1^{\mathtt{ih}} \in \mathcal{R}\langle \iota^{\mathtt{ih}}{}_{\vec{\mathsf{y}}@u}, \sigma^{\mathtt{ih}} \rangle$. Consequently, we conclude that $r_1^{\mathtt{ih}} \in \mathcal{R}\langle \iota^{\mathtt{ih}}{}_{\vec{\mathsf{y}}@u}, \sigma^{\mathtt{ih}}[x \mapsto \mathtt{true}] \rangle$ holds. Moreover,

$$\langle \iota(\mathsf{q}), \sigma \rangle \xrightarrow{\vdash \mathtt{true}} \langle \overline{\mathsf{y}_j}\,\mathsf{v}; \iota^{\mathtt{ih}}(\mathsf{q}), \sigma[x \mapsto \mathtt{true}] \rangle \xrightarrow{\vdash \overline{\mathsf{y}_j}\mathsf{v}} \langle \iota^{\mathtt{ih}}(\mathsf{q}), \sigma[x \mapsto \mathtt{true}] \rangle$$

Hence, $\langle \mathsf{q}, \overline{\mathsf{y}_j}\mathsf{d} \rangle r_1^{\mathtt{ih}} \in \mathcal{R}\langle \iota_{\vec{\mathsf{y}}@u}, \sigma \rangle$ which covers $\langle \mathsf{q}, \overline{\mathsf{y}_j}\mathsf{d} \rangle r_1$ by the inductive hypothesis.

- [RTCom$_2$] The proof follows analogously to the previous case.
- [RTSeq] Then $\Delta = \Delta_1; \Delta_2$ and $r = r_1 r_2$ with $r_1 \in \mathcal{R}_{\vec{\mathsf{y}}}(\Delta_1)$ and $r_2 \in \mathcal{R}_{\vec{\mathsf{y}}}(\Delta_2)$. Note that Δ_1 and Δ_2 are viable, otherwise Δ would not be viable. Given the structure of Δ, the typing of P can be achieved with an application of [VSeq], [VLoop], or [VSend].
- [VSeq] We have $P = P_1; P_2$; then there are specifications Δ_1' and Δ_2' such that $e \lrcorner \Gamma \ \vdash\ P_i \ \triangleright\ \Delta_i'$ for $i \in \{1, 2\}$, $\Delta' = \Delta_1'; \Delta_2'$. We consider two cases depending on whether $\Delta_1' = (\Delta_0')^\star$ for some Δ_0'.

  When there is no Δ_0' such that $\Delta_1' \neq (\Delta_0')^\star$ we proceed as follows. For $i \in \{1, 2\}$, by inductive hypothesis on $r_i \in \mathcal{R}_{\vec{\mathsf{y}}}(\Delta_i')$ we know that there are $\iota_i^{\mathtt{ih}}{}_{\vec{\mathsf{y}}@u}$, $\Gamma_i^{\mathtt{ih}} \supseteq \Gamma$, and $\Delta_i^{\mathtt{ih}} = (\Delta_{i,\mathsf{r}})_{\mathsf{r} \in \mathcal{P}(\mathcal{G})}$ compatible with e, Γ, and Δ_i'. Namely,
  (1) $\iota_i^{\mathtt{ih}}(\mathsf{p}) = P_i$ and
  (2) $e \lrcorner \Gamma_i^{\mathtt{ih}} \ \vdash\ \iota_i^{\mathtt{ih}}(\mathsf{r}) \ \triangleright\ \Delta_{i,\mathsf{r}}$ and $\Im\big(\Delta_{i,\mathsf{r}}(\vec{\mathsf{y}},\mathsf{r})\big) = \Im\big(\Delta_i'(\vec{\mathsf{y}},\mathsf{r})\big)$, for all $\mathsf{r} \in \mathcal{P}(\mathcal{G})$w

(3) for all $\sigma_i^{\mathtt{ih}} \ltimes (\Delta_i^{\mathtt{ih}}; e; \iota_i^{\mathtt{ih}}{}_{\vec{\mathsf{y}}@u}; \Gamma_i^{\mathtt{ih}})$ there is $r_i^{\mathtt{ih}} \in \mathcal{R}\langle \iota_i^{\mathtt{ih}}{}_{\vec{\mathsf{y}}@u}, \sigma_i^{\mathtt{ih}}\rangle$ such that $r_i \lessdot r_i^{\mathtt{ih}}$.
Now let

$$\begin{array}{rcll} \iota(\mathsf{r}) & = & \iota_1^{\mathtt{ih}}(\mathsf{r}); \iota_2^{\mathtt{ih}}(\mathsf{r}) & \forall \mathsf{r} \in \mathcal{P}(\mathcal{G}) \\ \iota(\mathsf{y}) & = & \iota_1^{\mathtt{ih}}(\mathsf{y}) & \forall \mathsf{y} \in \vec{\mathsf{y}} \\ \Gamma' & = & \Gamma_1^{\mathtt{ih}} \cup \Gamma_2^{\mathtt{ih}} & \\ \Delta' & = & \Delta_1'; \Delta_2' & \\ \sigma & = & \sigma_1^{\mathtt{ih}} & \end{array}$$

and observe that Δ' is defined otherwise compatibility would be violated, contradicting the inductive hypothesis.

If $\Delta_1' = (\Delta_0')^\star$ for some Δ_0', we note that $\Delta_1 = (\Delta_0)^\star$ for some Δ_0 because they are the same when restricted to participants' session of p. Then, r_1 has been obtained by using either [RTIt$_1$] or [RTIt$_2$]. Consequently, either $r_1 \in \mathcal{R}_{\vec{\mathsf{y}}}(\Delta_0')$ or $r_1 = r_0[r_3]$ with $r_0 \in \mathcal{R}_{\vec{\mathsf{y}}}(\Delta_0')$.

We consider $r_1 \in \mathcal{R}_{\vec{\mathsf{y}}}(\Delta_0')$ (the other case follows analogously). Since $\Delta_1' = (\Delta_0')^\star$ and $e \lrcorner \Gamma \vdash P_1 \triangleright (\Delta_0')^\star$, $P_1 = \mathsf{for}\ x\ \mathsf{in}\ \ell\ \mathsf{do}\ P_0$ and $e \lrcorner \Gamma \vdash P_0 \triangleright \Delta_0'$. As before, we apply inductive hypothesis on Δ_0, Δ_0', P_0, and r_o; we hence obtain a system $\iota_1^{\mathtt{ih}}{}_{\vec{\mathsf{y}}@u}$, an environment $\Gamma_1^{\mathtt{ih}}$, and a specification $\Delta_1^{\mathtt{ih}}$ such that $\iota_1^{\mathtt{ih}}(\mathsf{p}) = P_0$ and for all stores $\sigma_1^{\mathtt{ih}} \ltimes (\Delta_1^{\mathtt{ih}}; e; \iota_1^{\mathtt{ih}}{}_{\vec{\mathsf{y}}@u}; \Gamma_1^{\mathtt{ih}})$ there is $r_1^{\mathtt{ih}} \in \mathcal{R}\langle \iota_1^{\mathtt{ih}}{}_{\vec{\mathsf{y}}@u}, \sigma_1^{\mathtt{ih}}\rangle$ such that $r_1 \lessdot r_1^{\mathtt{ih}}$. Likewise, proceeding as in the case of [VSeq] above, there are a system $\iota_2^{\mathtt{ih}}{}_{\vec{\mathsf{y}}@u}$, an environment $\Gamma_2^{\mathtt{ih}}$, and a specification $\Delta_2^{\mathtt{ih}}$ such that $\iota_2^{\mathtt{ih}}(\mathsf{p}) = P_2$ and for all stores $\sigma_2^{\mathtt{ih}} \ltimes (\Delta_2^{\mathtt{ih}}; e; \iota_2^{\mathtt{ih}}{}_{\vec{\mathsf{y}}@u}; \Gamma_2^{\mathtt{ih}})$ there is $r_2^{\mathtt{ih}} \in \mathcal{R}\langle \iota_2^{\mathtt{ih}}{}_{\vec{\mathsf{y}}@u}, \sigma_2^{\mathtt{ih}}\rangle$ such that $r_2 \lessdot r_2^{\mathtt{ih}}$. The proof ends by taking

$$\begin{array}{rcll} \iota(\mathsf{p}) & = & P & \\ \iota(\mathsf{r}) & = & \mathsf{repeat}\ \iota_1^{\mathtt{ih}}(\mathsf{r})\ \mathsf{until}\ \iota_2^{\mathtt{ih}}(\mathsf{r}) & \forall \mathsf{r} \neq \mathsf{p} \in \mathcal{P}(\mathcal{G}) \\ \iota(\mathsf{y}) & = & \iota_1^{\mathtt{ih}}(\mathsf{y}) & \forall \mathsf{y} \in \vec{\mathsf{y}} \\ \Gamma' & = & \Gamma_1^{\mathtt{ih}} \cup \Gamma_2^{\mathtt{ih}} & \\ \Delta' & = & \Delta_1'; \Delta_2' & \\ \sigma & = & \sigma_1^{\mathtt{ih}} & \end{array}$$

The case $r_1 = r_0[r_3]$ with $r_0 \in \mathcal{R}_{\vec{\mathsf{y}}}(\Delta_0')$ is analogous.

- [VLoop] The thesis follows as in the previous case noticing that $P = \mathsf{repeat}\ M\ \mathsf{until}\ N$ (that is, P plays a passive role in the loop) and then using the inductive hypothesis on the premisses of the typing of P and observing that Δ is viable, hence there it yields an active role deciding when to terminate the loop.
- [VSend] The thesis follows trivially as in the cases [RTCom$_1$] and [RTCom$_2$] by observing that Δ' assigns to the participant session $(\vec{\mathsf{y}}, \mathsf{p})$ an output on a session channel $\mathsf{y} \in \vec{\mathsf{y}}$.

Note that [RTIt1] and [RTIt2] cannot be the last rules applied in a derivation of $r \in \mathcal{R}_{\vec{\mathsf{y}}}(\Delta)$. □

Theorem 8.8 (Typeability & coverage)**.** *Let $\mathcal{G}(\vec{\mathsf{y}}) \triangleq \mathsf{G}$ be a global type with $\mathcal{P}(\mathsf{G}) = \{\mathsf{p}_0, \ldots, \mathsf{p}_n\}$ and $\Delta = (\vec{\mathsf{y}},\mathsf{p}_0) : \mathbb{T}_0, \ldots, (\vec{\mathsf{y}},\mathsf{p}_n) : \mathbb{T}_n, \vec{\mathsf{y}} : []$ such that $\jmath(\mathbb{T}_i) = \mathsf{nf}(\mathsf{G}\upharpoonright \mathsf{p}_i)$ for all $0 \leq i \leq n$. If $e \lrcorner \Gamma \vdash P \triangleright \Delta', (\vec{\mathsf{y}},\mathsf{p}_i) : \mathbb{T}_i$ then the set $\mathbb{I} = \{\iota_{\vec{\mathsf{y}}@u} \mid e \lrcorner \Gamma \vdash \iota_{\vec{\mathsf{y}}@u} \triangleright \Delta, \Delta'' \wedge \iota(\mathsf{p}_i) = P\}$ covers $\mathcal{R}_{\vec{\mathsf{y}}}(\Delta)$, i.e., $\mathcal{R}_{\vec{\mathsf{y}}}(\Delta) \Subset \mathcal{R}\mathbb{I}$.*

Proof. The proof follows directly from Lemma B.14. □

Appendix C. Whole-spectrum implementations and Guarded Automata

In this section we briefly discuss how our notion of whole-spectrum implementation (WSI) can be defined when specifications and implementations are defined as Guarded Automata.

We first recall some basic definitions from [21]: (P, M) is a composition schema where $P = \{\mathsf{p}_1, \ldots, \mathsf{p}_n\}$ is a set of participants and M are the messages (i.e., the alphabet), $R = \langle (P, M), A \rangle$ is a conversation protocol where A is a guarded automaton, $W = \langle (P, M), A_1, \ldots, A_n \rangle$ is a web service composition, $L(R) = L(A)$ is the language of a conversation protocol. For a web service composition $W = \langle (P, M), A_1, \ldots, A_n \rangle$ we have runs, send sequences and conversations;

(1) a run of W is a sequence of configurations $\gamma = c_0, c_1, \ldots, c_n$ where:
   - c_0 is an initial configuration
   - $c_i \rightarrow c_{i+1} \quad (i = 0 \ldots n-1)$
   - c_n is a final configuration
(2) a send sequence γ on a run γ is the sequence messages, one for each send action in γ, recorded in the order in which they are sent,
(3) a conversation is a word w over M for which there is a run γ of W such that $w = \gamma$,
(4) the conversations of a web service W, written $C(W)$, is the set of all the conversations for W.

We are now ready to introduce a notion of WSI for guarded automata.

Definition C.1 (Whole-spectrum realisation of a guarded automaton)**.** Let P be a set of participants defined as $\{\mathsf{p}_1, \ldots, \mathsf{p}_n\}$. A_i is a whole-spectrum realisation of $\mathsf{p}_i \in P$ in conversation protocol $R = \langle (P, M), A \rangle$ if for all $w \in L(R)$ there exist $\{A_j\}_{j \in \{1,\ldots,n\} \setminus \{i\}}$ such that $w \in C(< (P, M), A_1, \ldots, A_n >)$.

Definition C.2 (WSI of guarded automaton)**.** A_i is a WSI of p_i in conversation protocol $R = \langle (P, M), A \rangle$ if: (1) A_i is a deterministic guarded automaton, and (2) A_i is a whole-spectrum realisation of p_i in $R = \langle (P, M), A \rangle$.